NASA-CR-196884

IN-60-CR

Generic POCC Architectures

Contract Number NAS5-31500
Task 28-11600

Prepared For:

Computer Sciences Corporation
8728 Colesville Road
Silver Spring, MD 20910

April 5, 1989

(NASA-CR-196884) GENERIC POCC ARCHITECTURES (Computer Technology Associates) 60 p

N95-11148

Unclas

G3/60 0023655

Prepared By:

CTA INCORPORATED
14900 Sweitzer Lane, Suite 201
Laurel, MD 20707
(301) 369-2400

INTRODUCTION

This document describes a generic POCC architecture based upon current POCC software practice, and several refinements to the architecture based upon object-oriented design principles and expected developments in teleoperations. The current-technology generic architecture is an abstraction based upon close analysis of the ERBS, COBE, and GRO POCCs. A series of three refinements is presented: these may be viewed as an approach to a phased transition to the recommended architecture. The third refinement constitutes the recommended architecture, which, together with associated rationales, will form the basis of the rapid synthesis environment to be developed in the remainder of this task.

The document is organized into two parts. The first part describes the current generic architecture using several graphical as well as tabular representations or "views." The second part presents an analysis of the generic architecture in terms of object-oriented principles. On the basis of this discussion, refinements to the generic architecture are presented, again using a combination of graphical and tabular representations.

CURRENT GENERIC ARCHITECTURE

This part of the document describes a generic POCC architecture based upon current software practice and control center operations. The architecture is essentially an abstraction of the ERBS, COBE, and GRO POCCs, based upon their common elements and features. In a few cases, a feature of one of these POCCs was selected as generic even though not shared by the other two POCCs. Such decisions were based upon extended discussion of POCC requirements and software design principles. Mappings of the generic architecture back to the three POCCs have been documented, so that it is clear what changes to the generic architecture would be required to develop an ERBS, COBE, or GRO. However, this material has not been included in this document because we understand our task as pointing to future systems rather than the past.

The current generic architecture is described using several representations or "views." The first view is a configuration table, which lists the subsystems of the POCC Applications Processor, and describes the placement of specific functions that could be placed elsewhere. Placement decisions that reflect current practice and that differ from the recommended generic architecture are indicated in boldface type.

The second view is a levelled set of entity-relationship (E-R) diagrams. The levelling reflects the fact that we have grouped together certain top-level subsystems found in ERBS, COBE, and/or GRO to form higher-level subsystems. The rationales for such grouping were 1) simplicity, and 2) common underlying data abstractions.

Following the E-R representation is a list of functions performed by each subsystem. The E-R view, together with this list, provides a convenient baseline for discussing alternative groupings and the functional adequacy of the POCC architecture.

The next set of diagrams presents a data flow view. Entities in this view correspond exactly to those in the E-R model, but instead of general relationships between entities, this view presents specific data flows. The data flow view is levelled in the same manner as the E-R view.

Finally, we present a dynamic view of the POCC architecture. A set of composition graphs describes the processing of the principal events that can arrive at, or occur within, the generic POCC. Processing of events may travel through several subsystems. Each processing box in the composition graphs is annotated with a subsystem descriptor. The descriptor identifies the subsystem that performs that processing step. Following the composition graphs, there is a textual description of the processing of each event. This text amplifies and explains the information that is represented concisely in the composition graphs.

POCC AP--GENERIC ARCHITECTURE CONFIGURATION

Subsystems

Offline	Separate
Operator Input and Display	Separate
History	Separate
External Simulator	Yes
CMS Interface	Separate

Functions

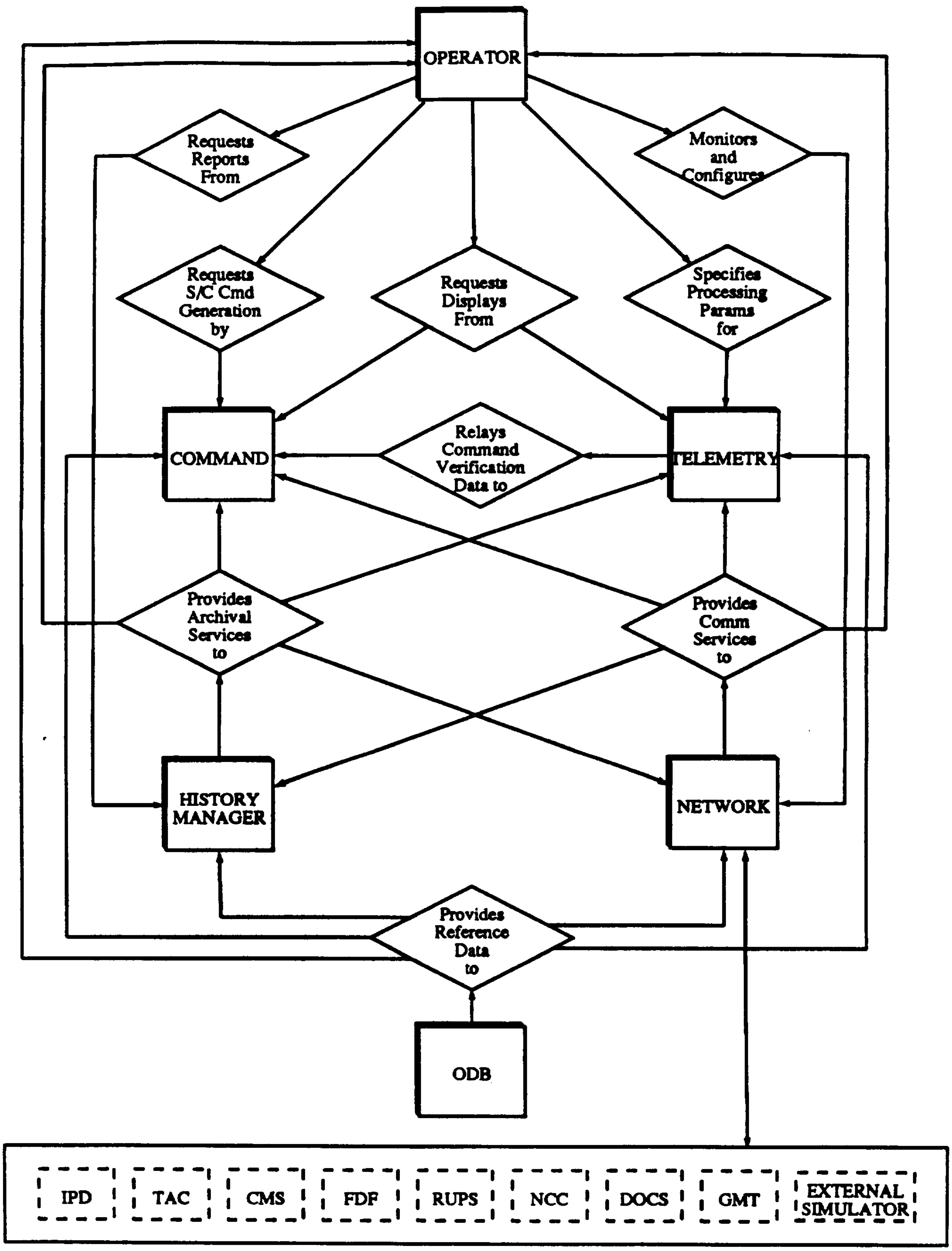
Initialization	Distributed
Database Access	**Distributed**
Database Files	Centralized
Internal Simulation	**Part of Telemetry**
External Simulator Control	**Local and Remote (AP) Operator**
Directive Processing	Distributed
Customer Directive Input	No
Single Interface for Operator	No
I/O Devices	
Device Control	**Distributed**
Knowledge Based UIF	No
Load/Image Verification	In Command
Telemetry Replay	In History
Non-Network Interface	**Distributed**
Network Interface	Centralized

CURRENT GENERIC ARCHITECTURE

ENTITY-RELATIONSHIP DIAGRAMS

NAL PAGE IS
OF POOR QUALITY

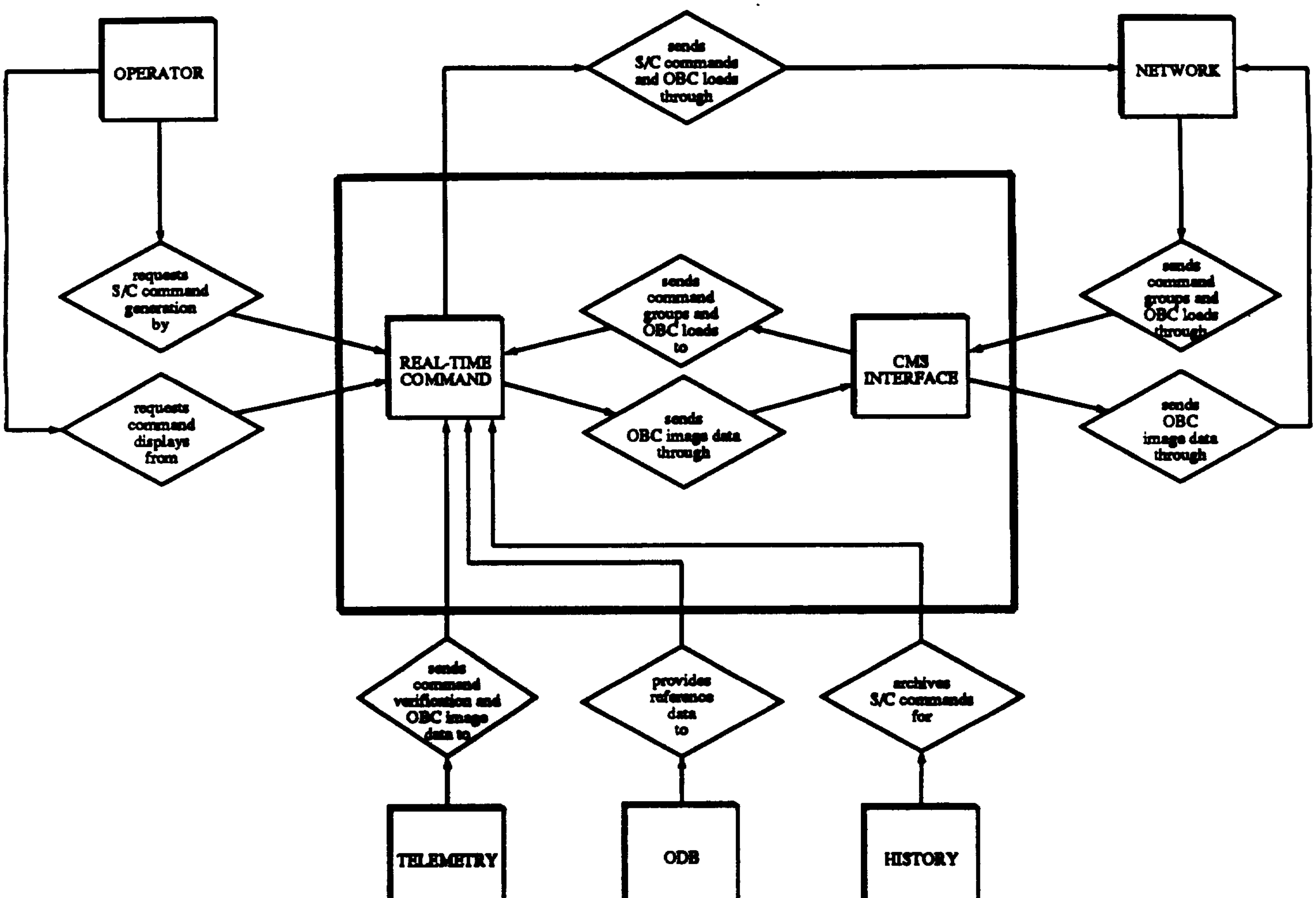
POCC AP--GENERIC ARCHITECTURE LEVEL 0



er0 3/28/89

POCC AP-GENERIC ARCHITECTURE

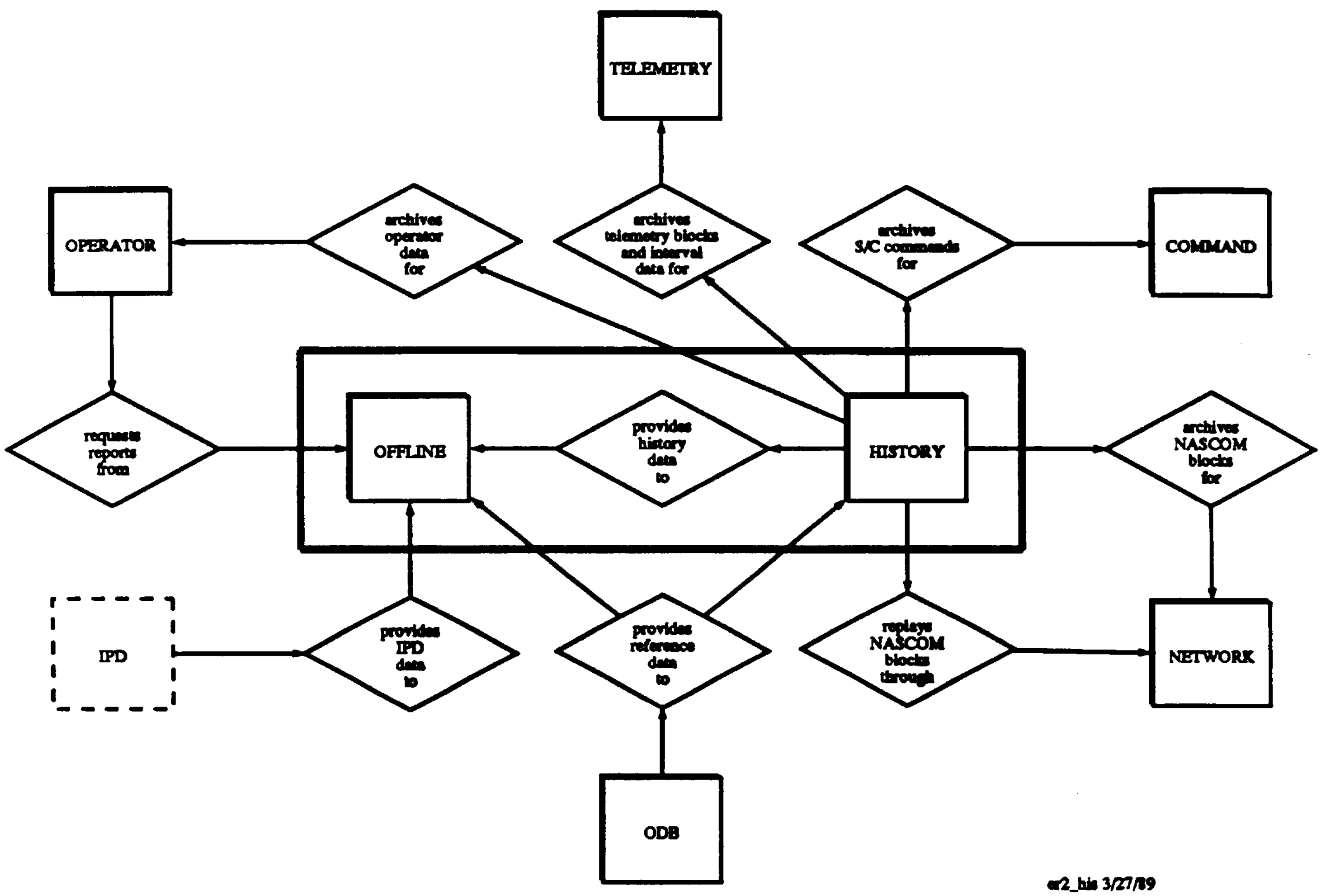
COMMAND



er2_com 3/28/89

POCC AP-GENERIC ARCHITECTURE

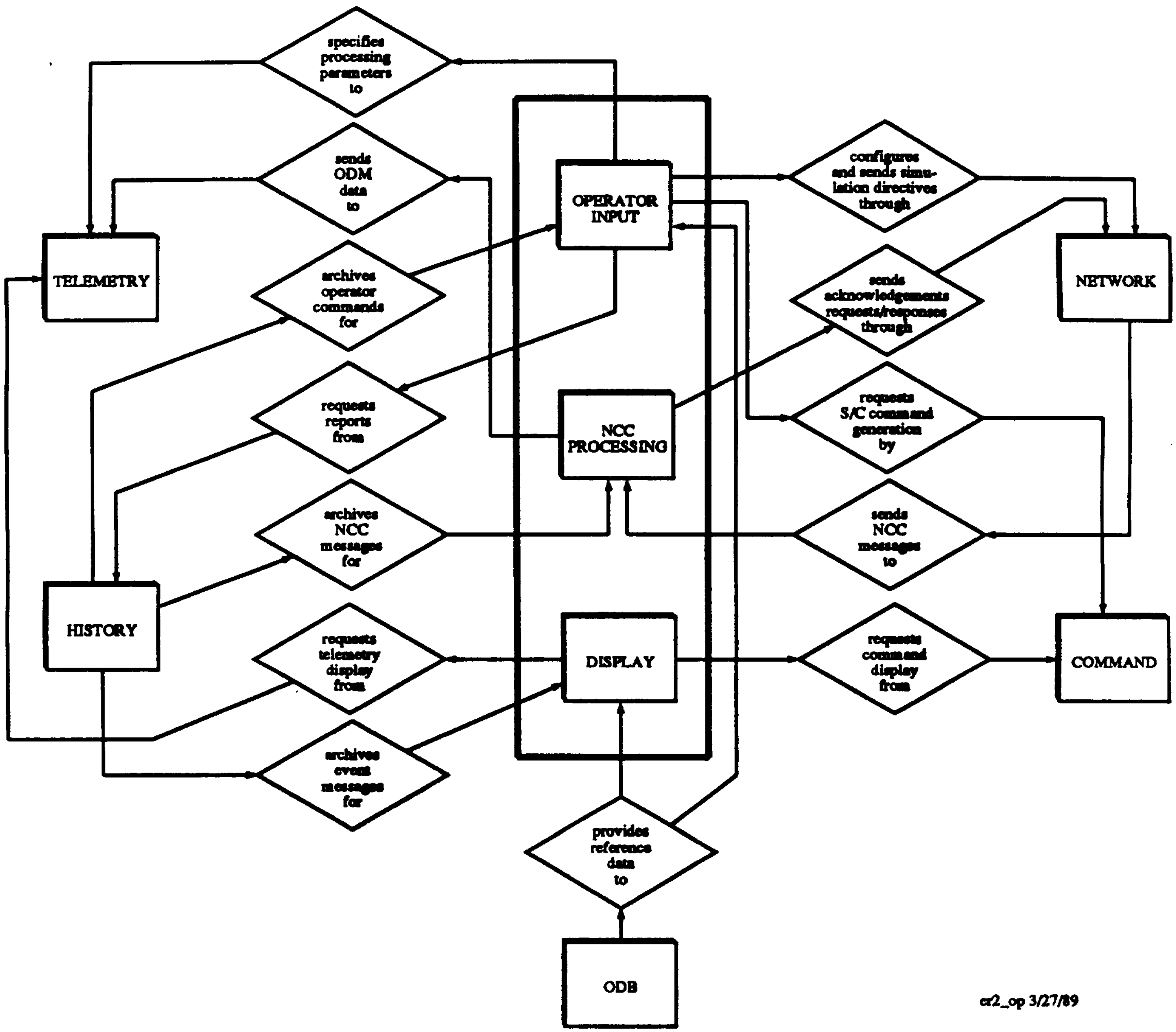
HISTORY MANAGER



er2_his 3/27/89

POCC AP-GENERIC ARCHITECTURE

OPERATOR



er2_op 3/27/89

March 29, 1989

GENERIC ARCHITECTURE 1--SUBSYSTEM FUNCTIONS

TELEMETRY

1. Interpret Operator Directives
2. Initialize Subsystem
3. Process telemetry blocks and route telemetry information to appropriate subsystems
4. Collect Interval Data
5. Provide Quick Look Experiments Analysis
6. Provide Special Algorithms (Equation Processors)
7. Check Analog Limits
8. Monitor Spacecraft Configuration
9. Provide Simulation
10. Copy Telemetry Blocks and Interval Data to History

DATA BASE

1. Initialize Subsystem
2. Create ODB from raw input
3. Update the ODB via Operator Input
4. Print out ODB

COMMAND.REAL-TIME COMMAND

1. Initialize Subsystem
2. Interpret Operator Directives
3. Receive Command Groups and OBC Loads from CMS Interface
4. Receive Images and Verification Data from Telemetry
5. Generate OBC Loads and Commands
6. Verify OBC Loads and Commands
7. Transmit OBC Loads and Commands to S/C via Network Interface
8. Send OBC Images to CMS Interface
9. Copy Commands to History

OPERATOR.NCC PROCESSING

1. Initialize Subsystem
2. Interpret Operator Directives
3. Parse Incoming NCC Blocks
4. Acknowledge NCC Block receipt
5. Generate NCC status requests/responses

6. Monitor NCC responses to status requests
7. Copy NCC Messages to History

OPERATOR.DISPLAY

1. Initialize Subsystem
2. Interpret and Process Operator Directives
3. Format Display Pages (Telemetry, SNAP, Special, etc.)
4. Drive External Devices
5. Manage Event Messages
6. Copy Event Messages to History

HISTORY MANAGER.HISTORY

1. Initialize Subsystem
2. Interpret Operator Directives
3. Record History Data from all subsystems
4. Generate Printer Listings for Event Messages, Commands, NASCOM Blocks, Tape Directory
5. Provide Telemetry and Interval Data for Offline Processing
6. Replay NASCOM Blocks to Telemetry via Network Interface

OPERATOR.OPERATOR INPUT

1. Parse and edit operator input (except DATABASE subsystem)
2. Verify User Access
3. Route directives to other subsystems
4. Process directives
5. Copy Operator Input to History

HISTORY-MANAGER.OFFLINE

1. Initialize Subsystem
2. Interpret Operator Directives
3. Perform S/C clock error calculation
4. Perform Long-term Trend Analysis
5. Summarize Out of Limit Data
6. Convert IPD History Tape to POCC AP Format
7. Download Interval Archival Data to Display (IDT)

NETWORK

1. Initialize Subsystem

2. Interpret and Process (Network configuration) Operator Directives
3. Format outgoing blocks
4. Route outgoing blocks to external entities
5. Strip incoming block headers
6. Route incoming blocks to appropriate subsystems
7. Copy NASCOM Blocks to History

COMMAND.CMS INTERFACE

1. Receive OBC Loads and Commands from CMS
2. Send OBC Loads and Commands to Command
3. Receive OBC image data from Command
4. Send OBC image data to CMS

EXTERNAL SIMULATOR

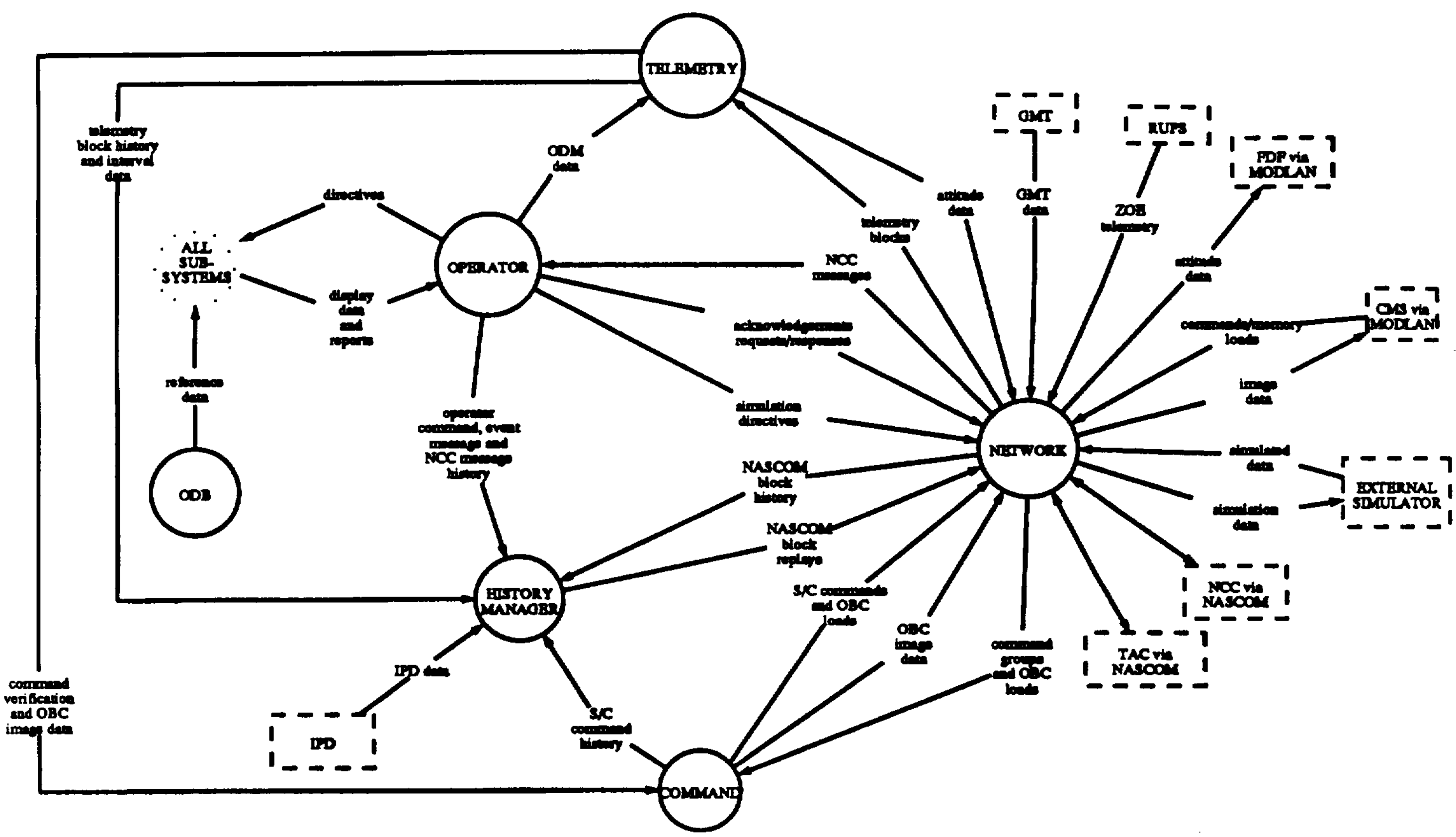
1. Initialize Subsystem
2. Interpret NASCOM Blocks
3. Interpret Directives
4. Execute S/C Commands
5. Generate Telemetry
6. Generate NCC Messages and Acknowledgements
7. Send Telemetry and NCC Messages/Acknowledegments to Network Interface

CURRENT GENERIC ARCHITECTURE

DATA FLOW DIAGRAMS

POCC AP--GENERIC ARCHITECTURE

DATAFLOW

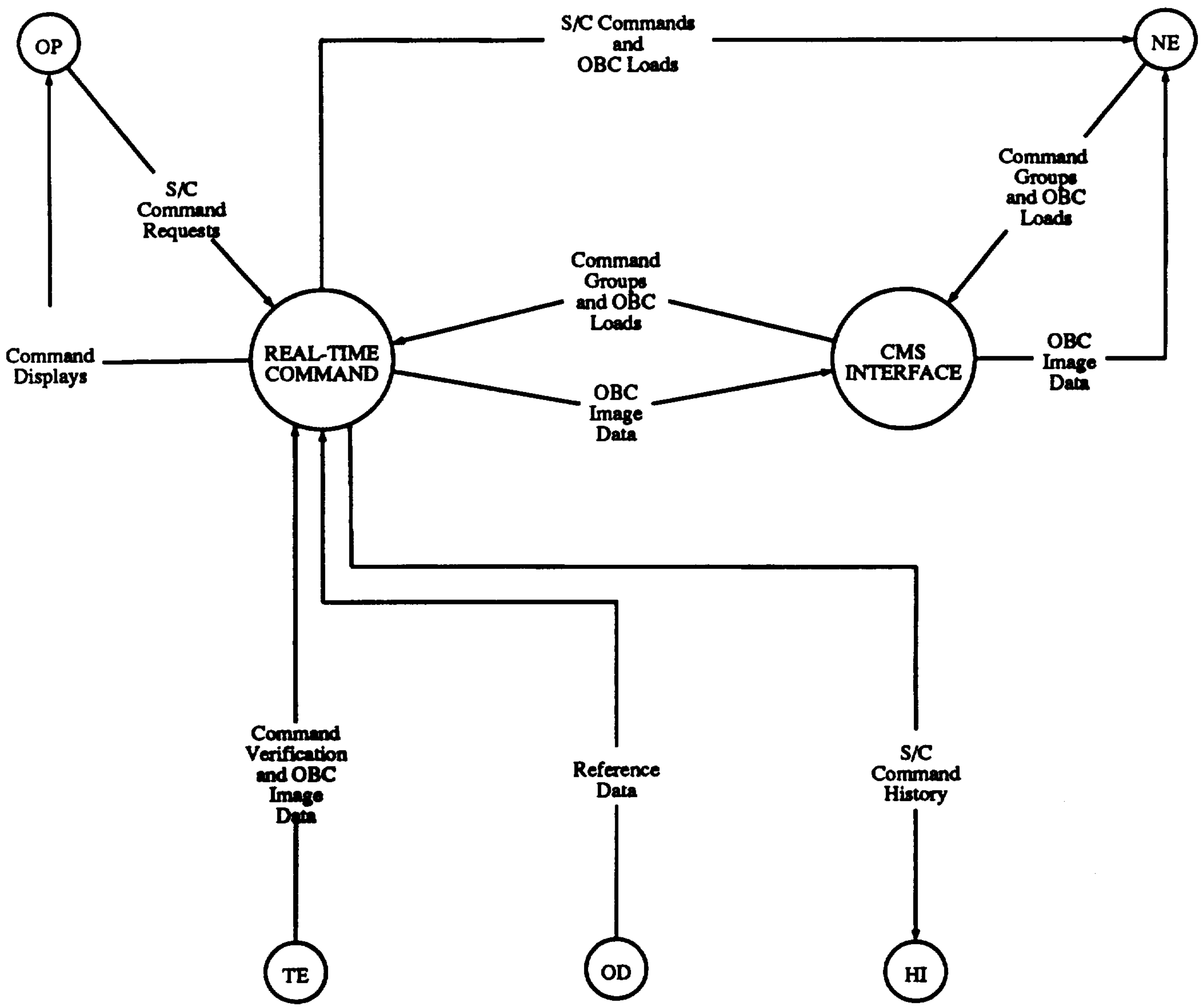


d30 3/30/89

POCC AP-GENERIC ARCHITECTURE

COMMAND

DATAFLOW

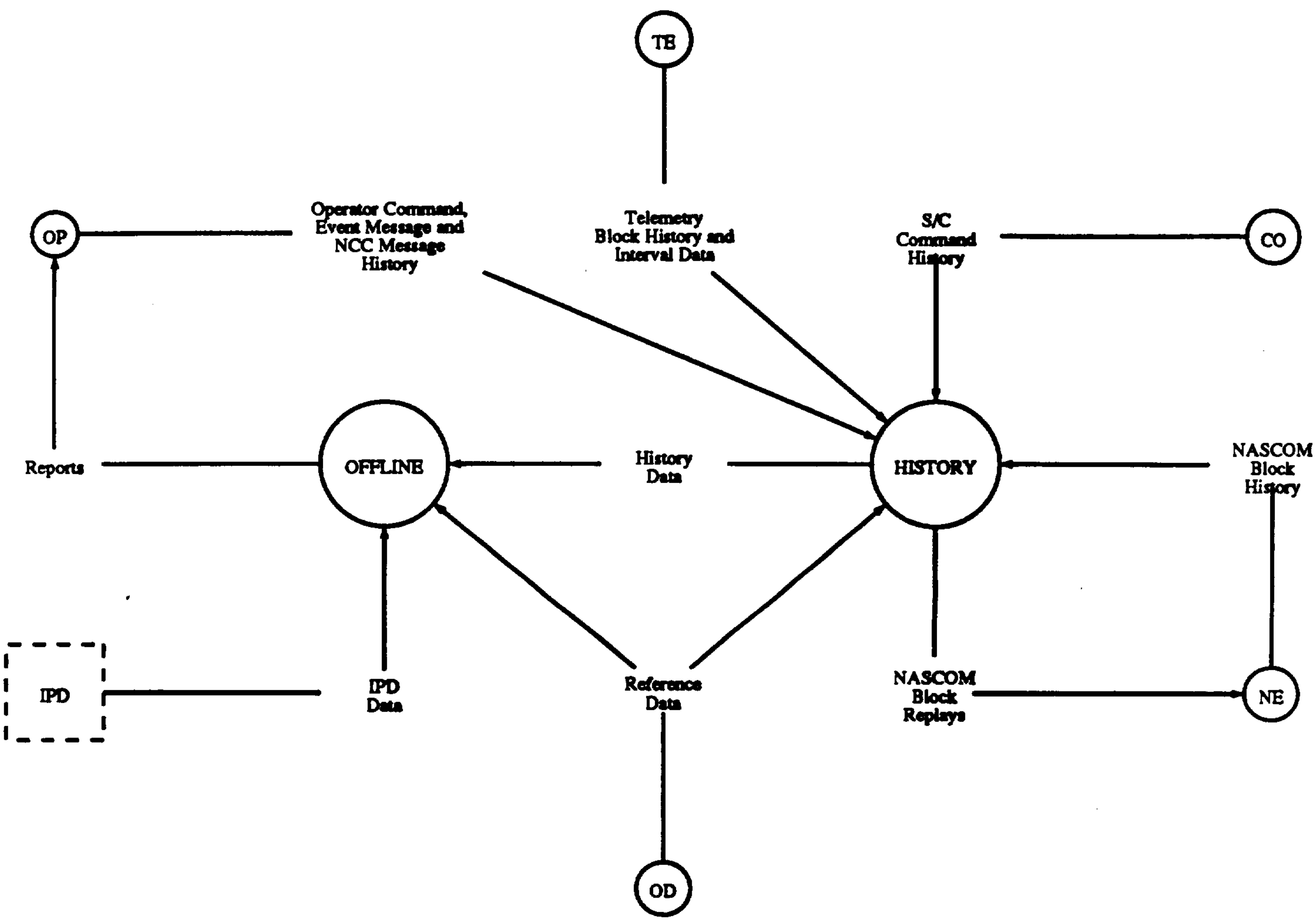


df2_com 3/30

POCC AP-GENERIC ARCHITECTURE

HISTORY MANAGER

DATAFLOW

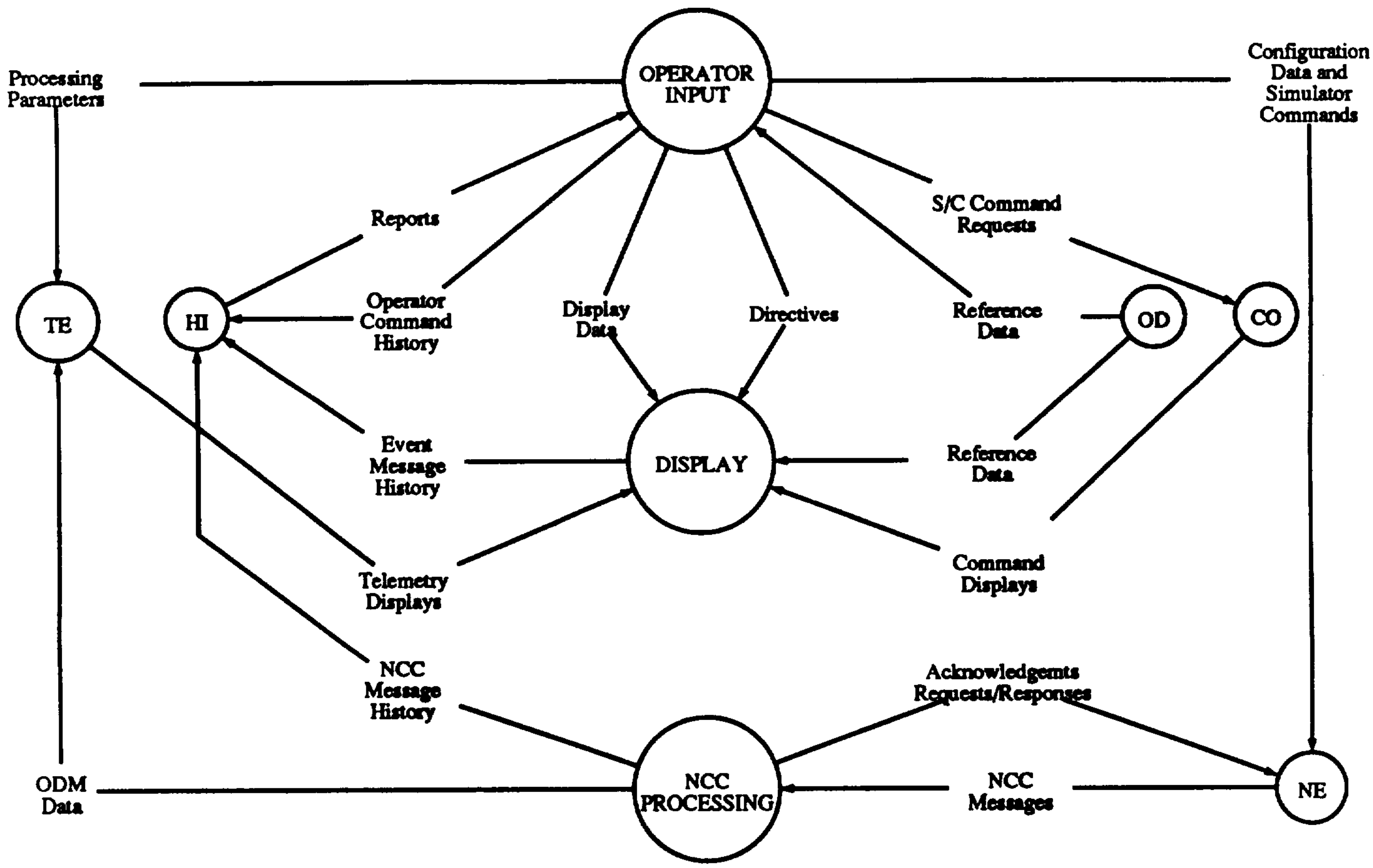


df2_his 3/28/89

POCC AP-GENERIC ARCHITECTURE

OPERATOR

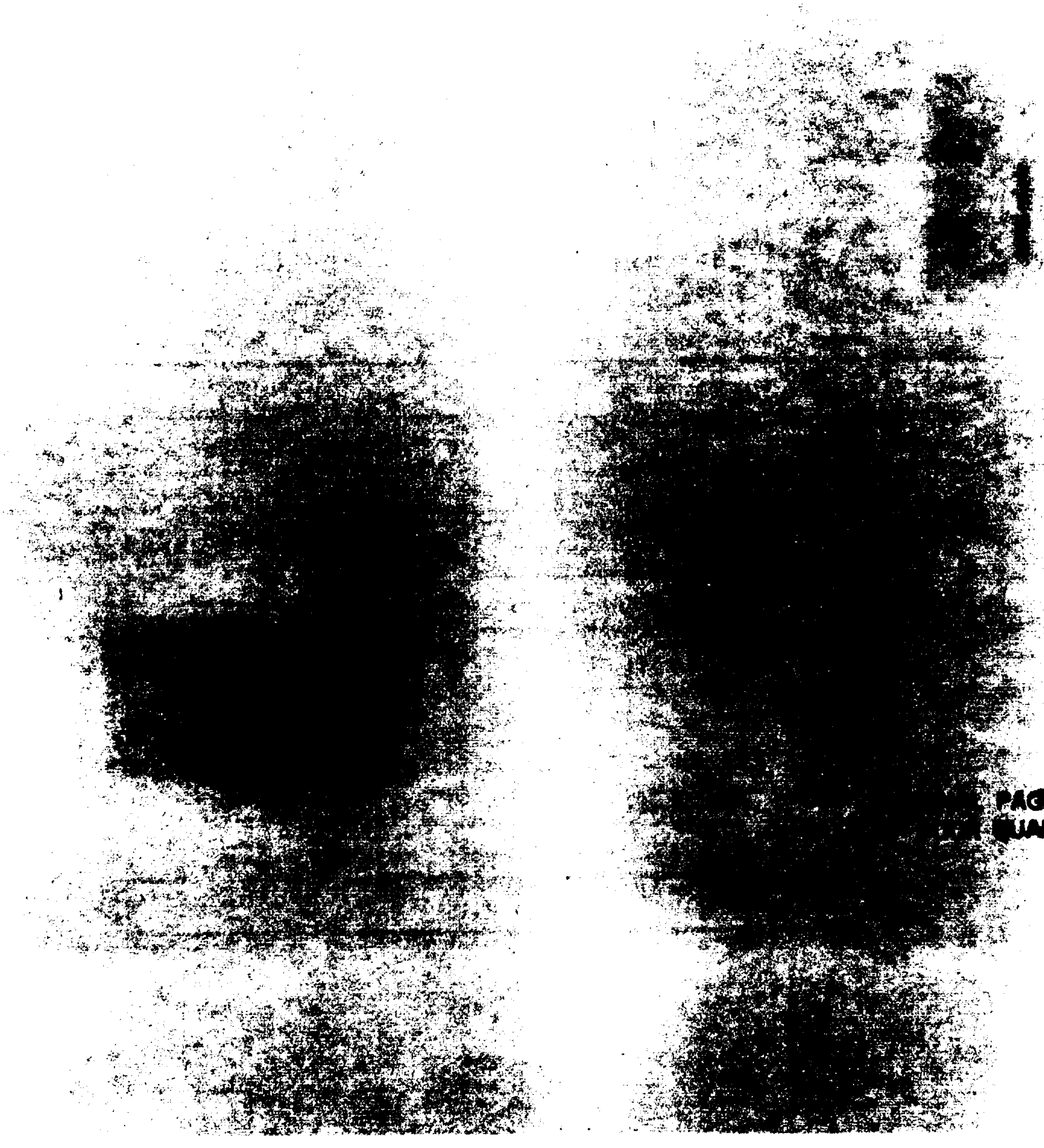
DATAFLOW



df2_op 3/30/89

CURRENT GENERIC ARCHITECTURE

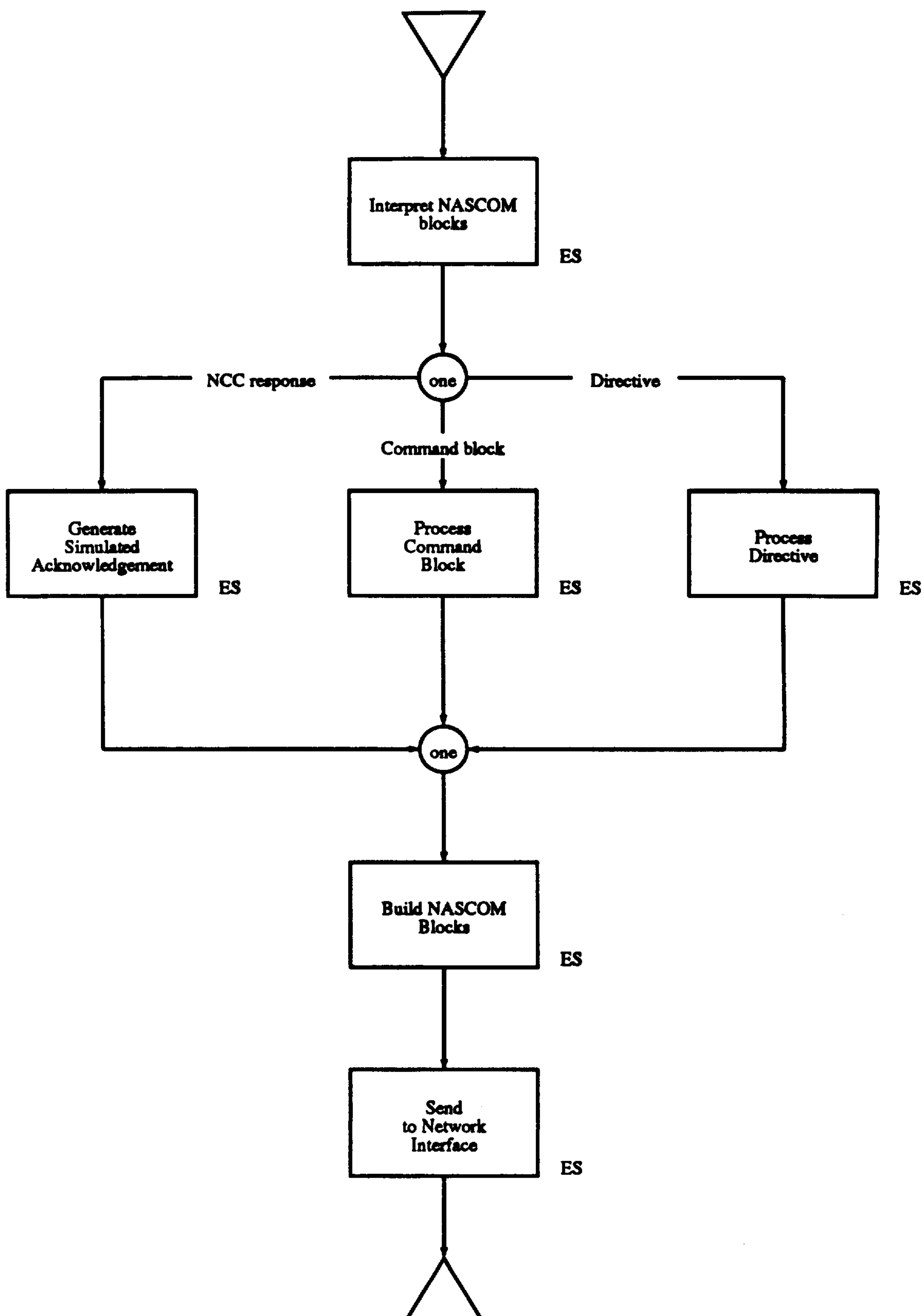
COMPOSITION GRAPHS



Legend for Composition Graphs

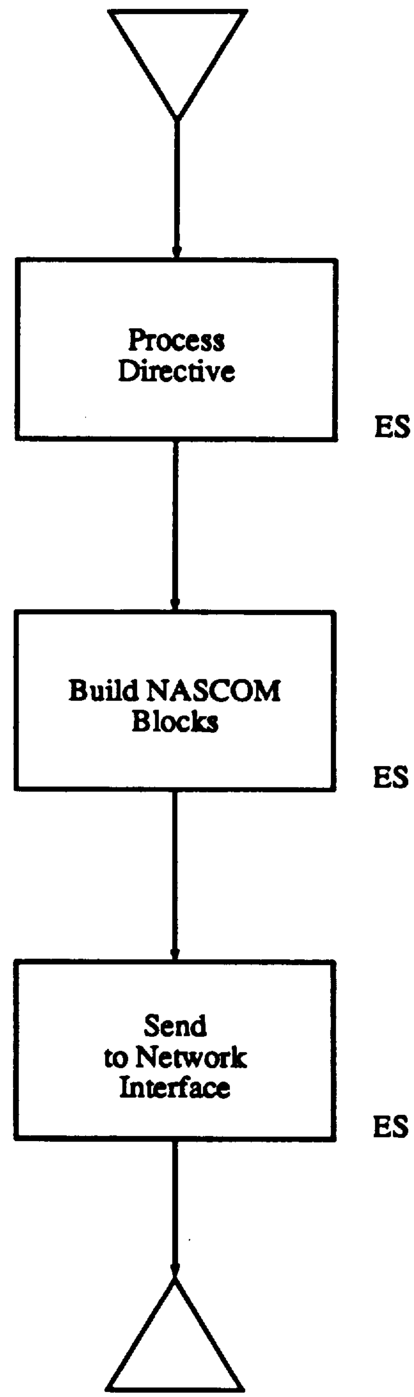
ES = External Simulator
DI = Operator.Display
HI = History Manager.History
OI = Operator.Operator Input
NI = Network
NP = Operator.NCC Processing
TE = Telemetry
CO = Command.Real-Time Command
CI = Command.CMS Interface

External Simulator Driven by NASCOM Blocks



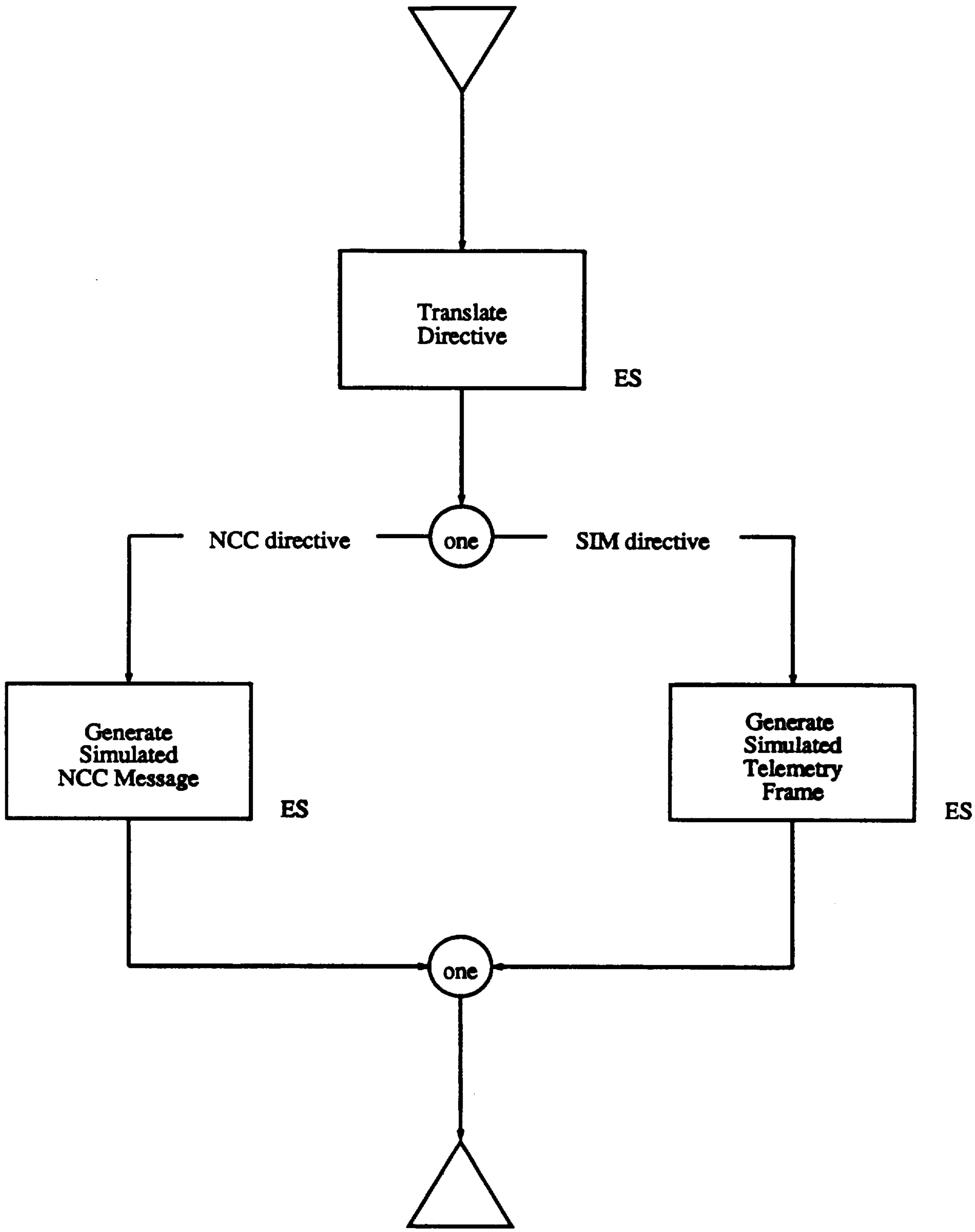
cg_es_nb 3/27/89

External Simulator Driven by Local Operator Directives



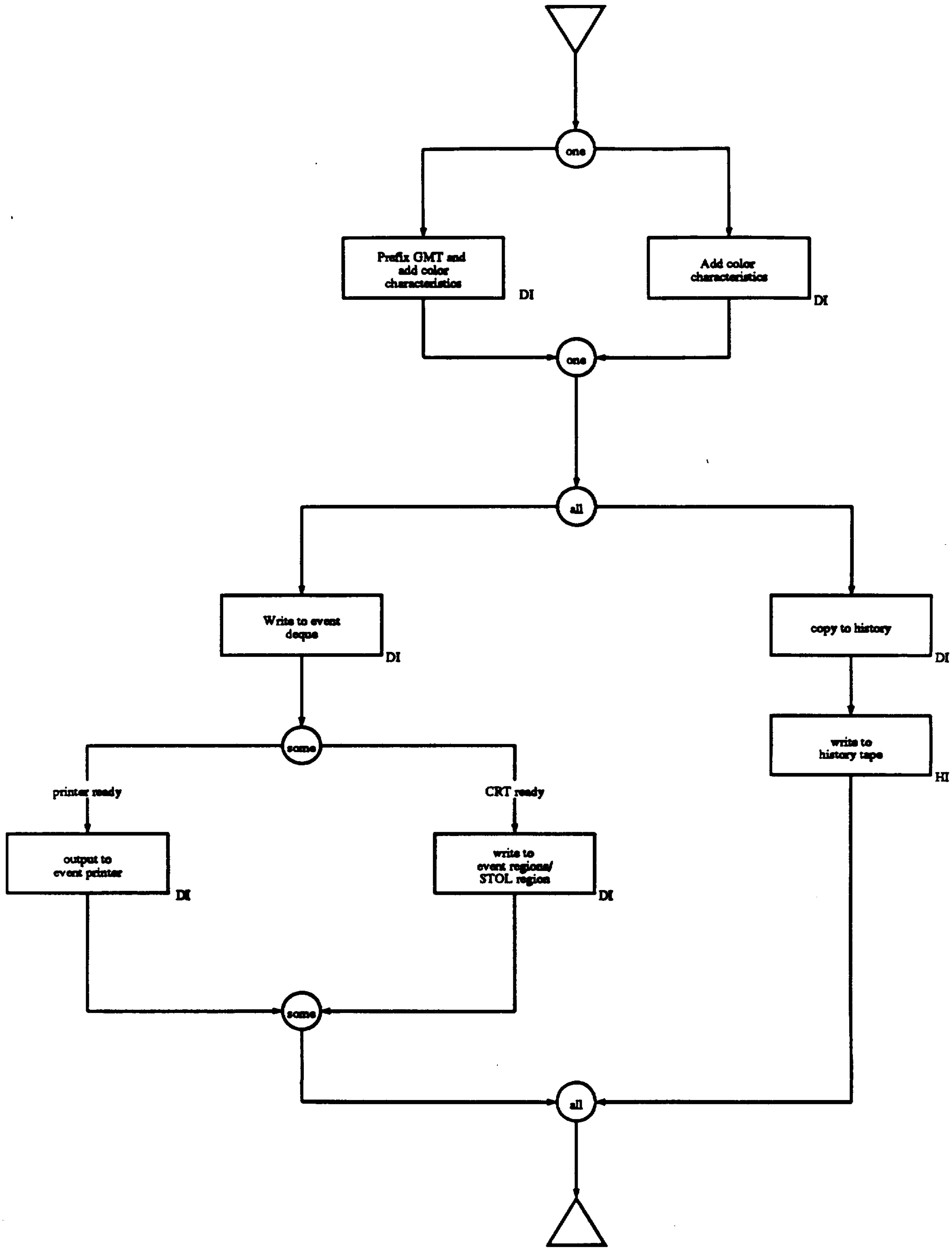
cg_se_op 3/27/89

Process Directive



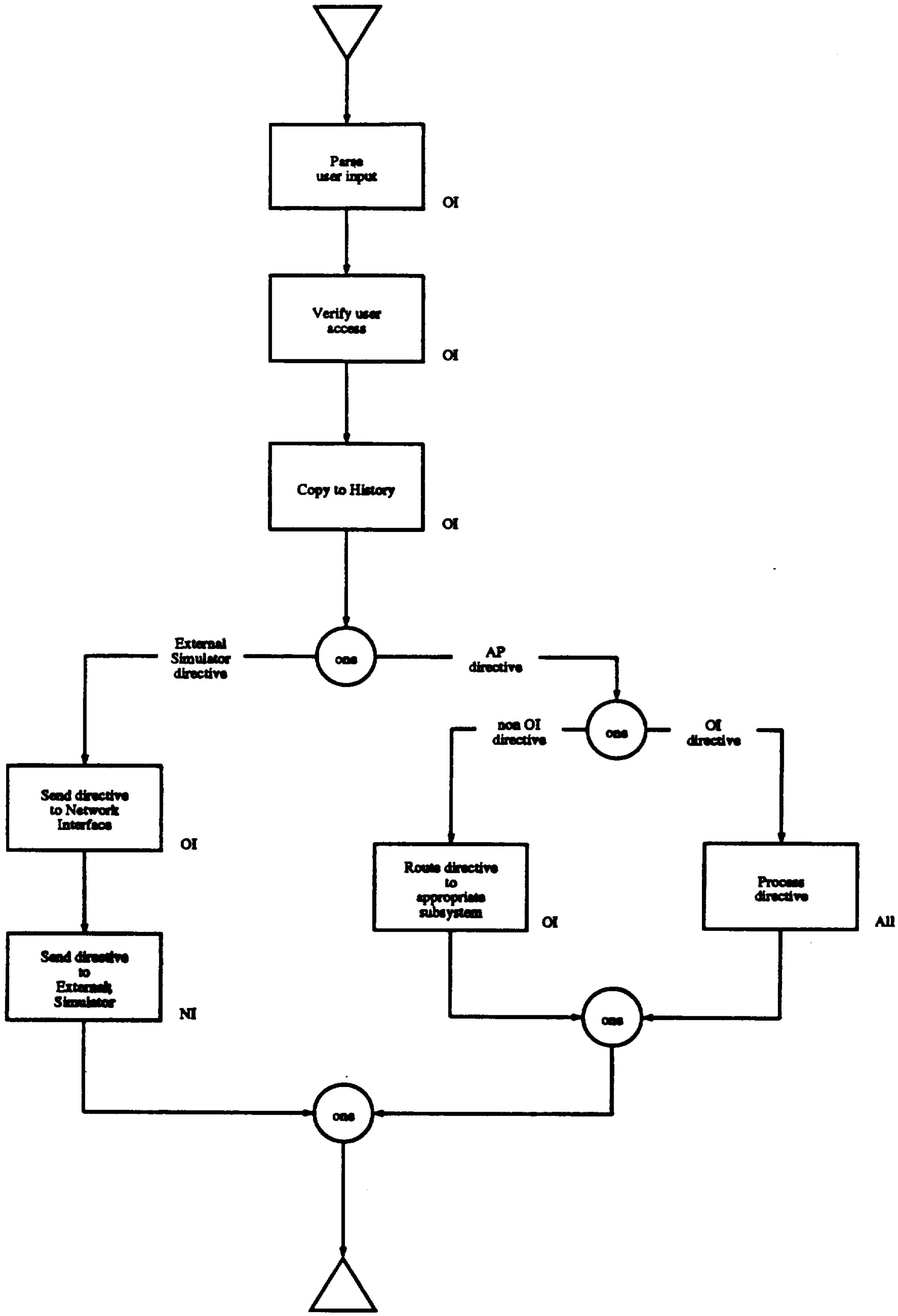
cg_es_dir 3/27/89

Process Event Message



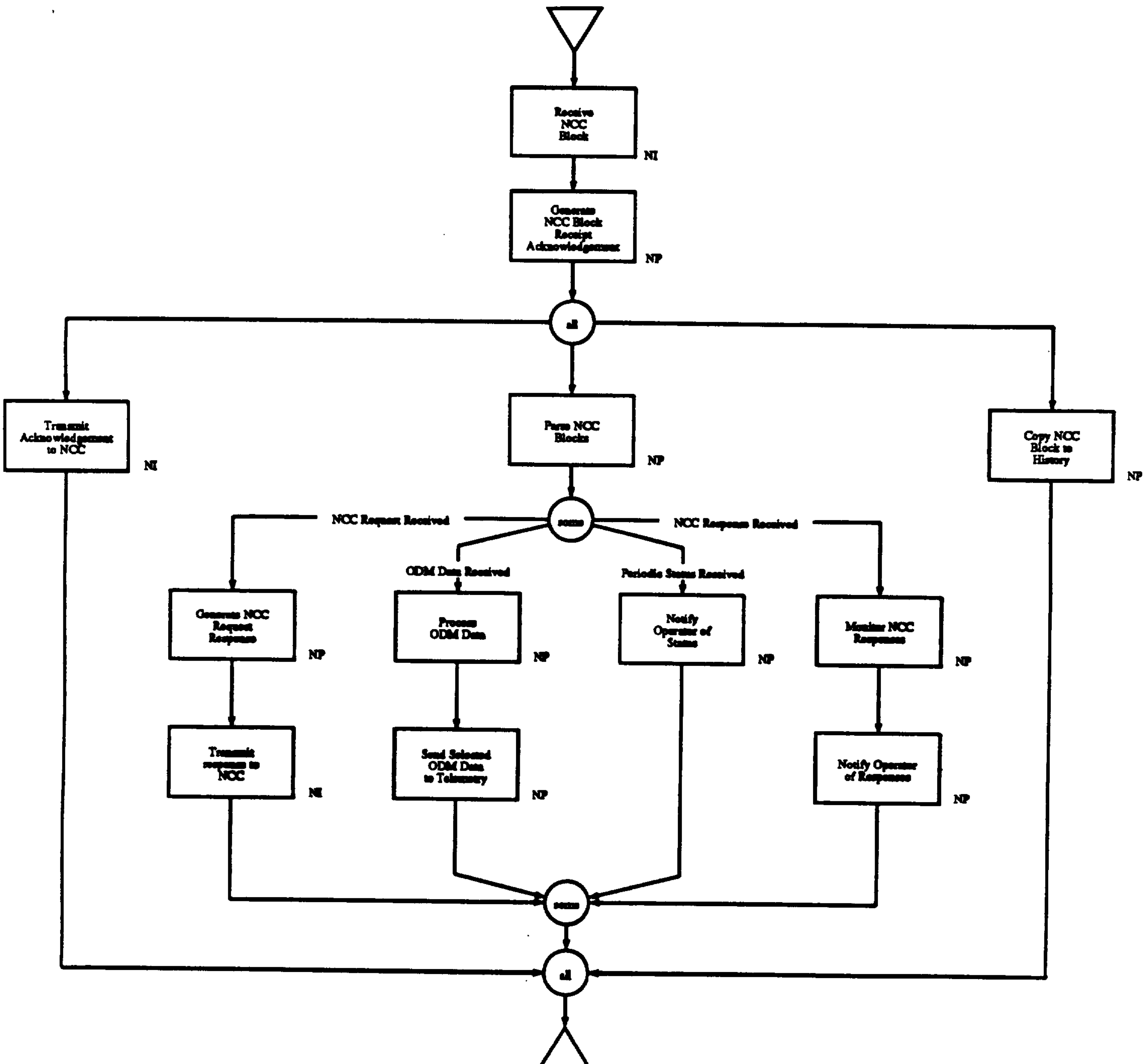
cg_ev 3/27/89

Process POCC AP Operator Directive



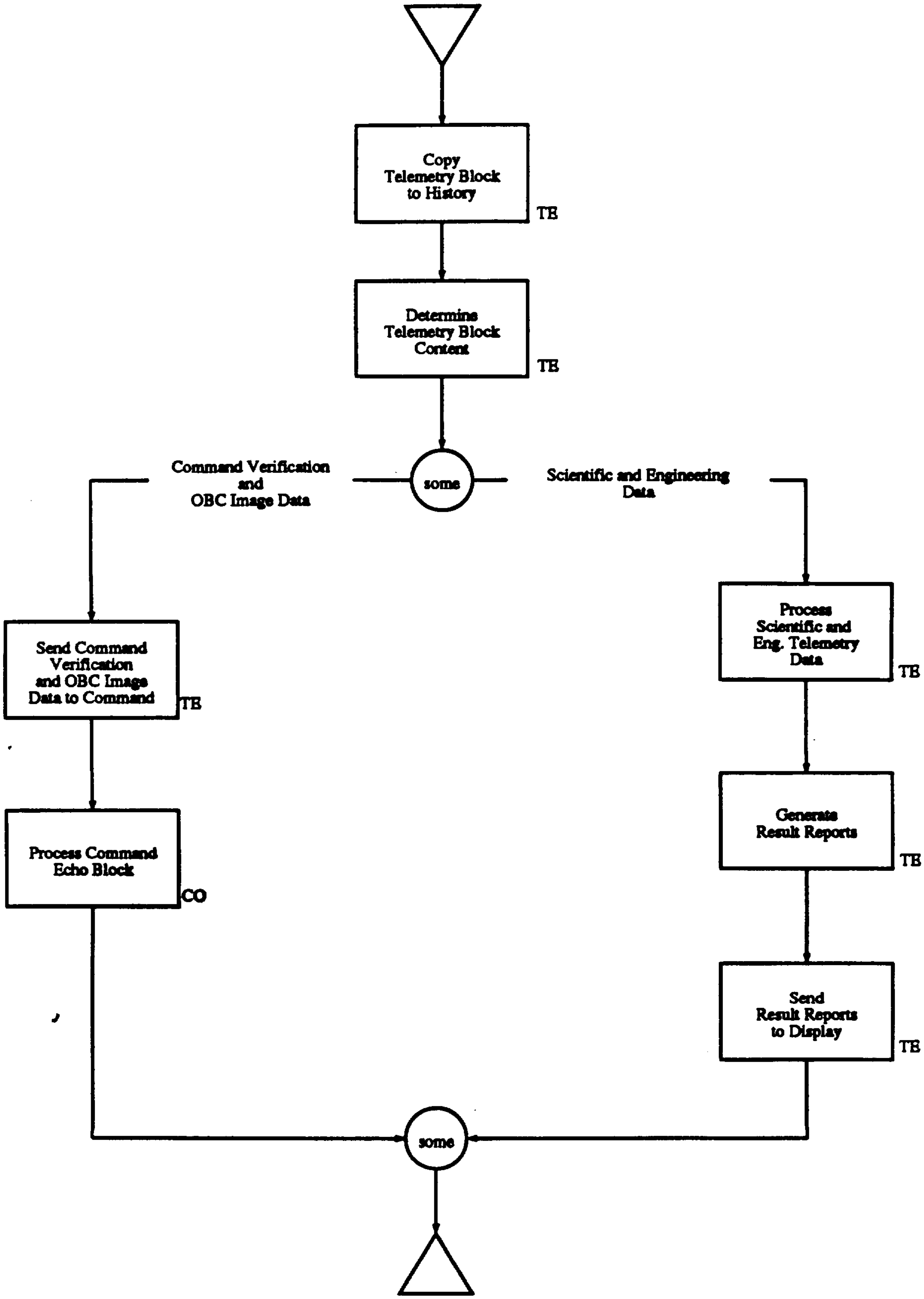
cg_op 3/27/89

Process NCC Block



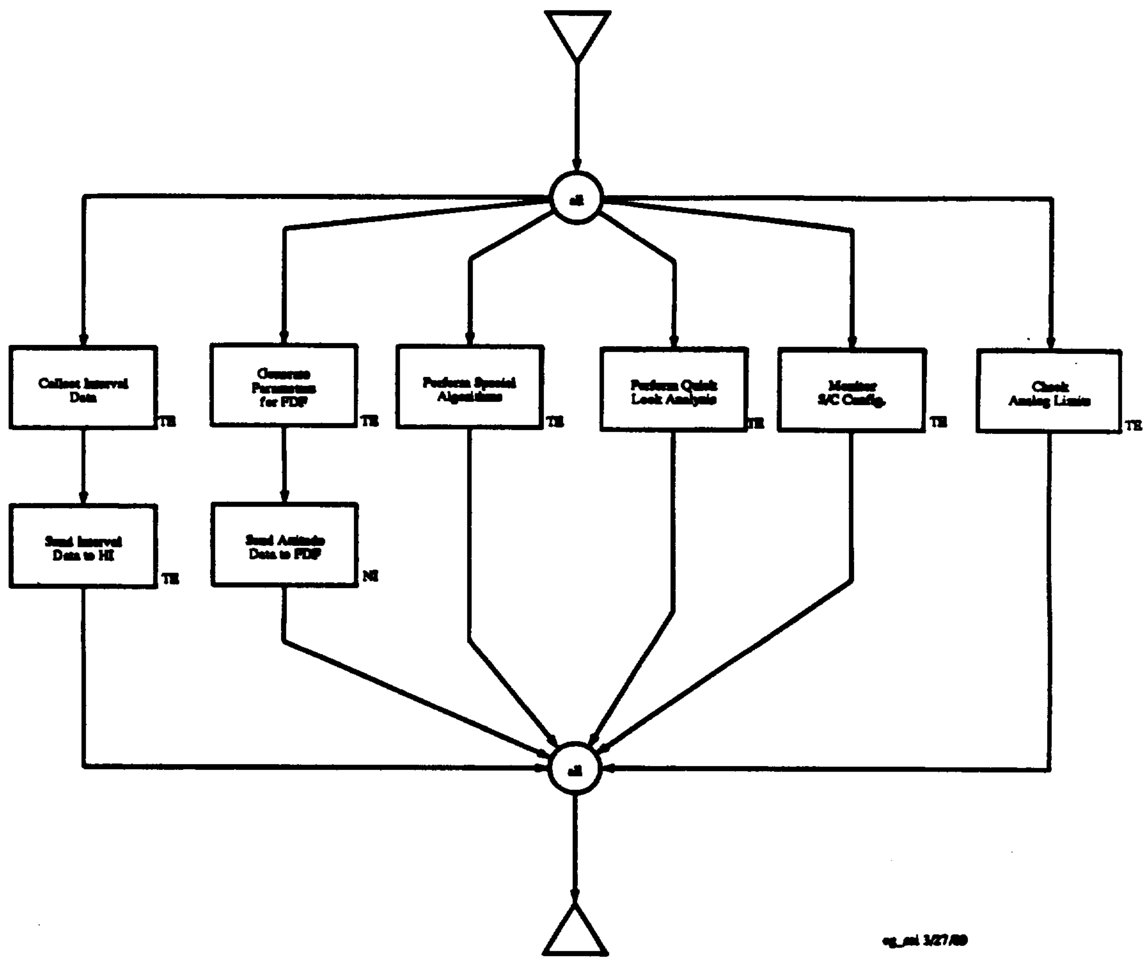
cg_nb 3/27/89

Process Telemetry Block



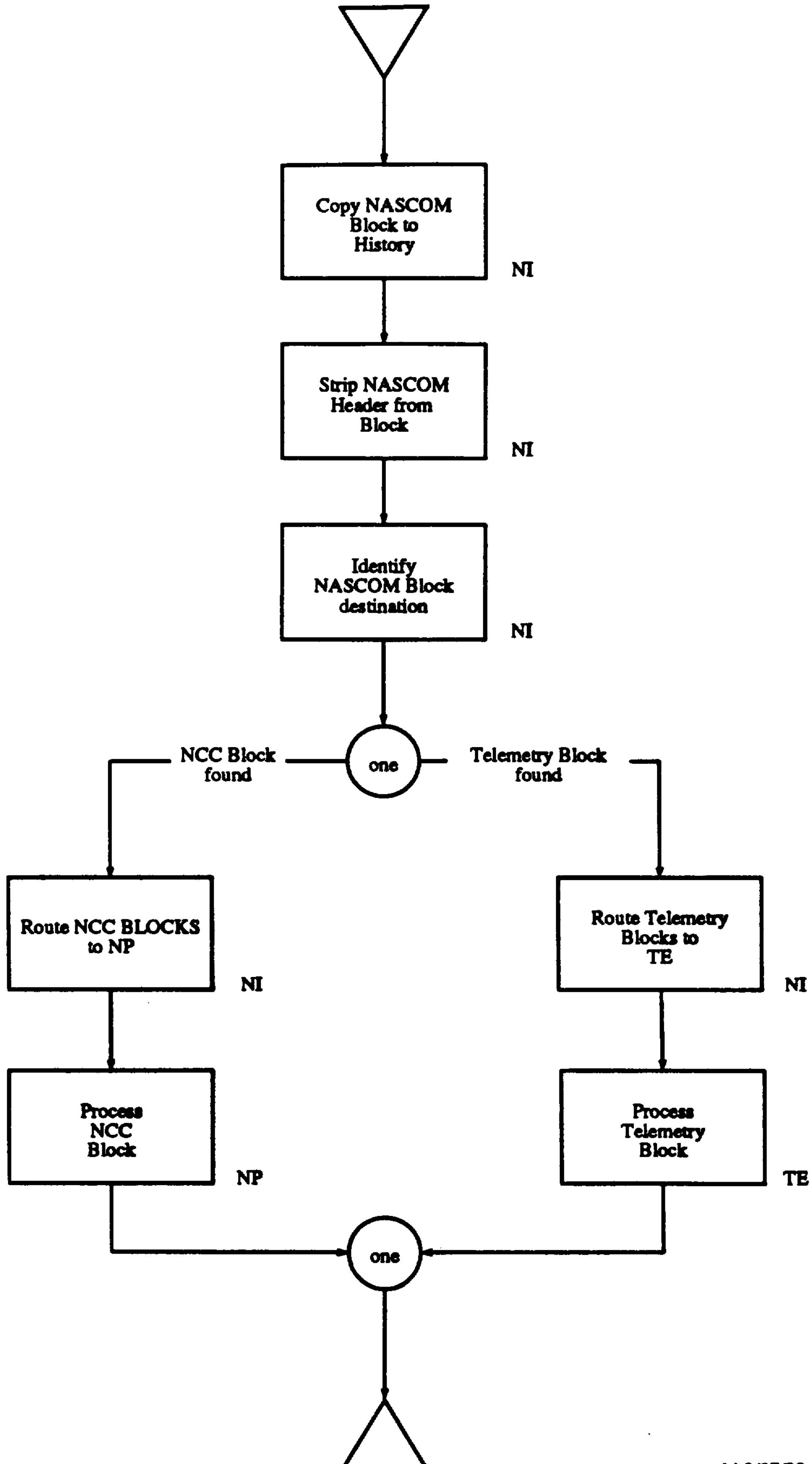
cg_te 3/27/89

Process Scientific and Engineering Telemetry Data



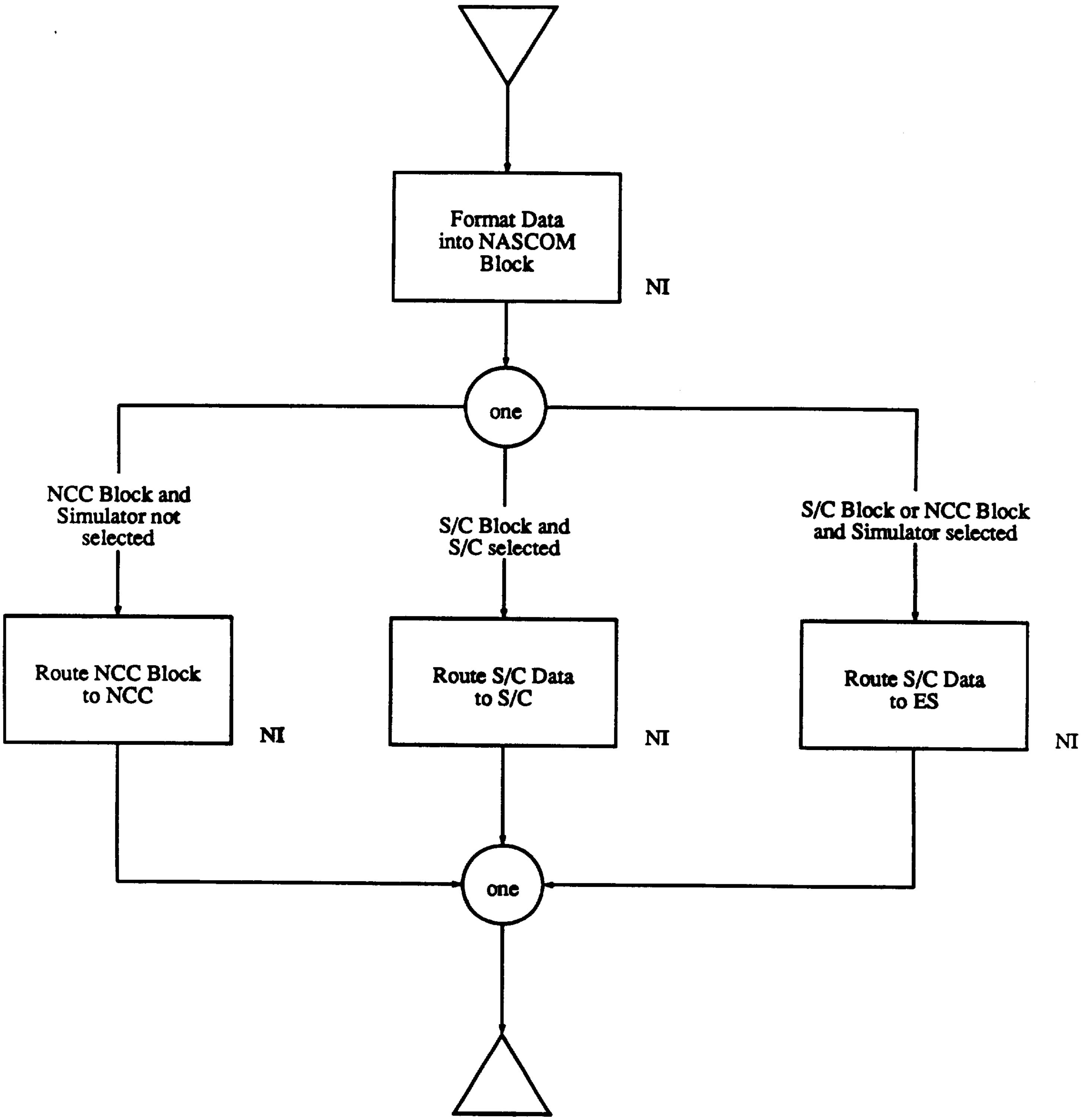
eg_sci 3/27/89

Process Incoming NASCOM Block



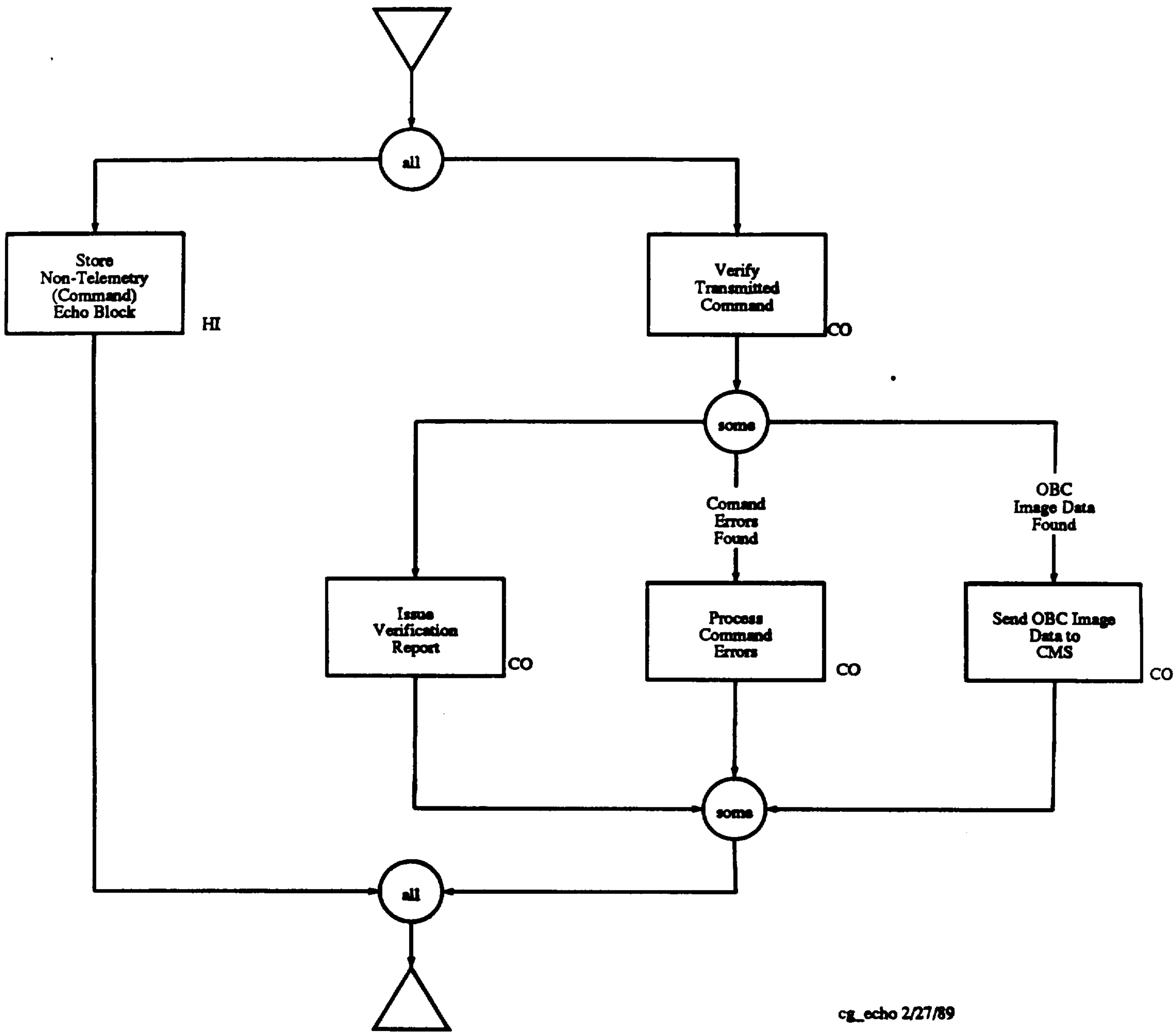
cg_nbi 3/27/89

Process Data to be Sent to External Systems via NASCOM



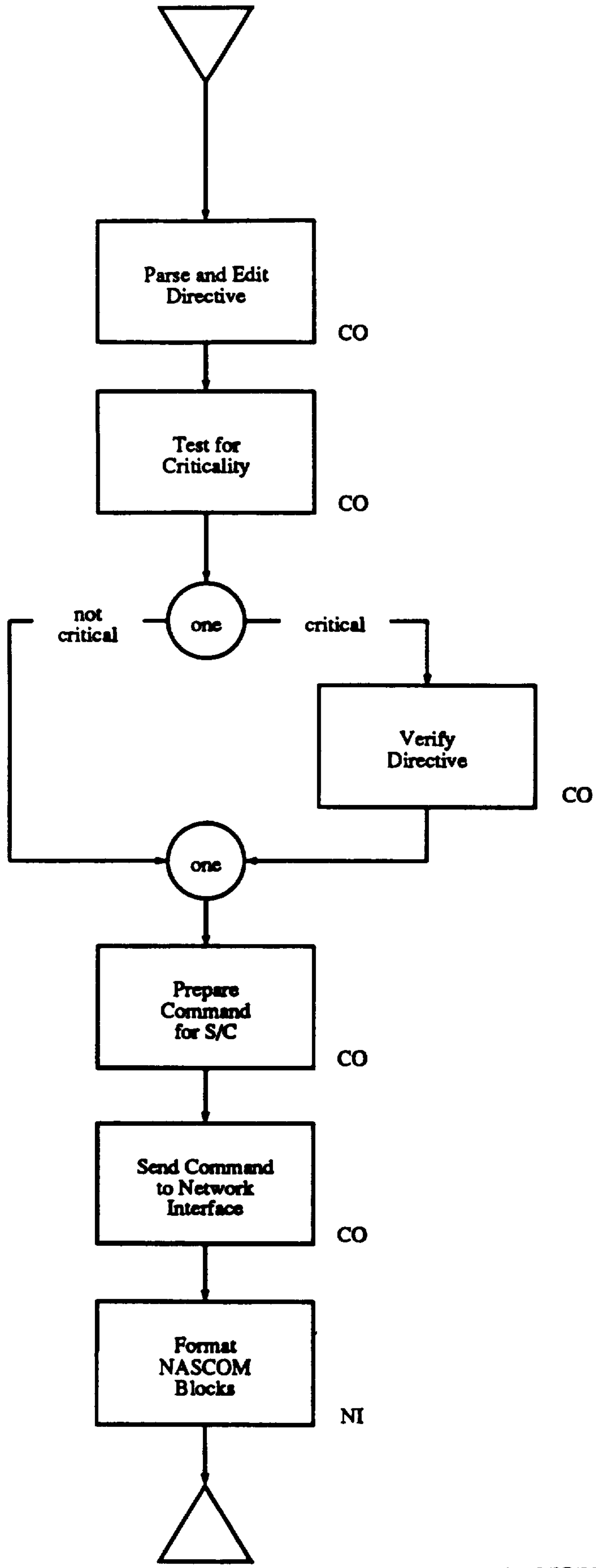
cg_2es 3/27/89

Process Command Echo Block



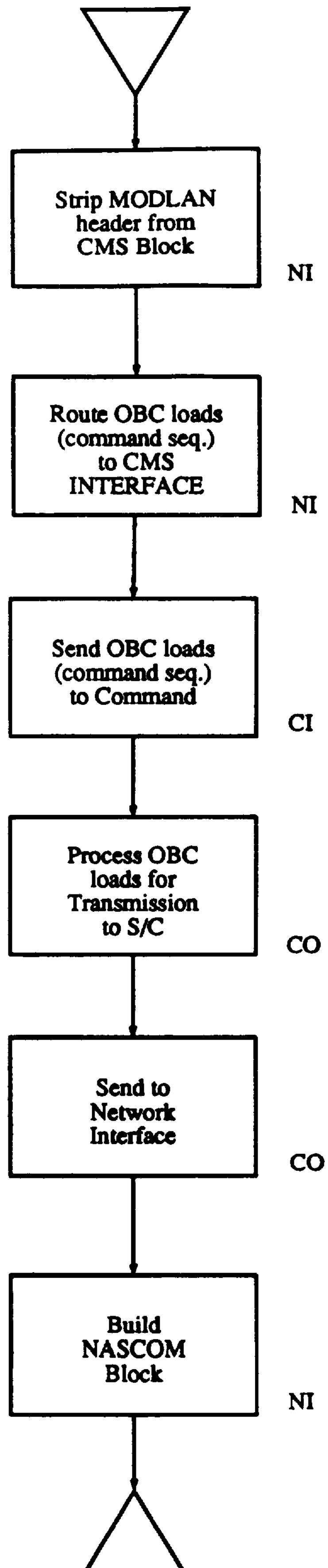
cg_echo 2/27/89

Process Operator Directive for S/C Command



cg_op4sc 3/27/89

Process OBC Loads (Command Sequences)



cg_obc 3/27/89

External Simulator

External Simulator Driven by NASCOM Blocks

The POCC AP communicates with the External Simulator through the POCC AP Network Interface Subsystem. Data are exchanged as NASCOM Blocks in both directions. These blocks may contain a directive originating from a remote operator, an NCC message response from the NCC Processing subsystem, or a S/C command from the Command subsystem. The External Simulator processes the directives and uses the NCC responses and Commands as input to the simulation. It generates simulated acknowledgements to NCC responses and simulates execution of S/C Commands. Simulation results are then formatted into NASCOM Blocks and sent via NASCOM to the NCC Processing or Telemetry subsystems of the POCC AP through its Network Interface subsystem.

Process Command Block

The External Simulator first examines command blocks to determine their content. SIM directives are used to drive the simulation, and Command data is used for simulating command execution.

External Simulator Driven by Local Operator Directives

The External Simulator may be controlled by directives from a local operator. These directives are processed in the same manner as those received as NASCOM Blocks from the POCC AP. Simulation results are formatted into NASCOM Blocks and routed through the POCC AP Network Interface subsystem to the NCC Processing or Telemetry subsystems.

Process Directive

Directives may come from a local operator or from a remote operator. If the directive was sent by a remote operator, it was sent through the POCC AP Network Interface subsystem which formatted the directive as a NASCOM block and sent it to the External Simulator via NASCOM. The External Simulator first interpretted this NASCOM block and then transferred it to this part of the system (see composition graph labeled "External Simulator Driven by NASCOM Blocks"). Directives are translated and categorized as either NCC directives, or SIM directives. SIM directives are used in generating simulated Telemetry Frames. NCC directives result in the generation of simulated NCC messages.

POCC AP

Process Event Message

All POCC AP subsystems create event messages and forward them to the Display subsystem for operator display. The Display subsystem determines whether the event messages may need to be prefixed (GMT) and/or have color characteristics added before they are displayed. The Display subsystem then copies the event message to the History subsystem which writes the event message to the appropriate history tape. The Display subsystem also writes the event message to the event deque. The event message may then be output to the specific event printer and/or to the specific event region/STOL region on the operator's CRT.

Process POCC AP Operator Directive

Operator directives are input to the POCC AP Operator Input subsystem. The Operator Input subsystem parses the user input and then verifies the user access. The directive is then sent to the History subsystem which maintains a record of operator directives. Based on the type of directive, the Operator Input subsystem then sends the directive to the appropriate POCC AP subsystem if it is an internal directive; if the directive is targeted to a system that is external to the POCC AP, it is sent to the Network Interface. The Network Interface then formats the directive according to the appropriate protocol and routes it to the specified system.

Process Data to be Sent to External Systems via NASCOM

All communications to systems external to the POCC AP are sent through the POCC AP Network Interface subsystem. The Network Interface subsystem formats the data into blocks using the appropriate protocol. Data which is to be output to external systems accessable via NASCOM are formatted into NASCOM blocks. The Network Interface then routes these NASCOM blocks to the appropriate external system based on the type of data and the selected subsystem. S/C Blocks contain S/C commands and OBC loads from the Command subsystem to be used by the spacecraft or for simulation. NCC Blocks originate from the NCC Processing subsystem. They may contain acknowledgements and responses to messages received from the NCC via NASCOM through the Network Interface subsystem, or, as a result of operator directives, they may contain requests to the NCC for information.

Process NCC Block

The NCC sends information via NASCOM to the POCC AP. This information first arrives at the POCC AP Network Interface subsystem formatted as a NASCOM Block. The Network Interface subsystem strips the NASCOM header information and routes the block to the NCC Processing subsystem. NCC Processing then generates an acknowledgement that it has received the NCC Block which is then forwarded to the Network Interface subsystem for transmitting to the NCC via NASCOM. NCC Processing also copies the NCC block to the History subsystem for later replay, and responds to the NCC Block based on its contents.

Process Telemetry Block

The POCC AP Network Interface subsystem receives telemetry formatted as a NASCOM Block from the TAC via NASCOM. The Network Interface subsystem strips the NASCOM header information and routes the block to the POCC AP Telemetry subsystem for processing. The Telemetry subsystem copies the block to the History subsystem for later replay, and processes the block based on its contents. The telemetry block may contain non-telemetry command echo blocks and/or OBC images which the Telemetry subsystem routes to the Command subsystem for further processing.

Scientific and engineering telemetry data are analyzed, and result reports are generated. These reports are sent to the Display subsystem for operator display.

Process Command Echo Block

Command responses and OBC Image Data are generated by the spacecraft and transmitted to the TAC in addition to the scientific and engineering telemetry. The TAC transmits the command responses and OBC Image Data together with the telemetry to the POCC AP via NASCOM. This information first arrives at the POCC AP Network Interface subsystem formatted as a NASCOM Block. The Network Interface subsystem strips the header information and routes the Telemetry block to the Telemetry subsystem. The Telemetry subsystem separates the non-telemetry data (command responses and OBC Image Data) and sends this to the History subsystem for storage as a non-telemetry echo block. The Telemetry subsystem also routes this non-telemetry (command) echo block to the Command subsystem for further processing. The Command subsystem uses the command echo block to verify that the OBC loads and commands were transmitted correctly. If any errors in transmission are detected, the Command subsystem forwards the original OBC load and command to the Network Interface for retransmission. The Command subsystem then summarizes the results of the command verification and sends a report to the Display subsystem for operator display. In addition, the Command subsystem forwards the OBC Image Data to the POCC AP CMS Interface subsystem. The CMS Interface subsystem then forwards the OBC Image data to the Network Interface for communication to the CMS.

Process Scientific and Engineering Telemetry Data

Scientific and Engineering Telemetry Data is processed and analyzed by the POCC AP Telemetry subsystem. This information may include interval data, data for quick look analyses, analog limits, and S/C configuration data. Interval data is collected and forwarded to the History subsystem for later replay. The Telemetry subsystem also performs several functions using the telemetry data as input. It does a quick look analysis as well as performing other special purpose algorithms, checks analog limits, monitors the S/C configuration, and generates parameters for the attitude data which is sent to the FDF via the Network Interface. The Telemetry subsystem will then use the results of its analyses and calculations to generate telemetry data summary reports which are forwarded to the Display subsystem for operator display.

Process Incoming NASCOM Block

All network communications from systems external to the POCC AP are first received by the Network Interface subsystem. Many of these communications are via NASCOM and arrive at the POCC AP Network Interface subsystem formatted as NASCOM Blocks. Upon receipt, the Network Interface first copies the NASCOM Block to the History subsystem. It then strips the NASCOM header from the block and determines which POCC AP subsystem the information should be sent to based on the type of block. NCC Blocks are routed to the NCC Processing subsystem, and Telemetry Blocks are routed to the Telemetry subsystem.

Process Operator Directives for S/C Command

An operator directive may be targeted for the spacecraft. This type of directive is sent by the Operator Input subsystem to the Command subsystem. The Command subsystem parses and edits the directive, and tests it for criticality. The Command subsystem then uses the directive to prepare the command(s) for the S/C. If the directive is determined to be critical, its use is first verified with the operator. The prepared S/C command(s) are then forwarded to the Network Interface subsystem. The Network Interface formats the S/C command as a NASCOM block and sends it via NASCOM to the TAC. The TAC would then transmit the command to the spacecraft.

Process OBC Loads (Command Sequences)

OBC loads (command sequences) originate from the CMS. The CMS sends the OBC loads (command sequences) to the POCC AP via MODLAN. The CMS Block arrives at the POCC AP Network Interface subsystem. The Network Interface subsystem first strips the MODLAN header from the CMS Block. It then routes the OBC loads (command sequences) to the POCC AP CMS Interface subsystem. The CMS Interface sends this information to the Command subsystem. The Command subsystem then processes the OBC loads for transmission to the spacecraft. The prepared OBC loads (command sequences) are then forwarded to the Network Interface. The Network Interface formats the OBC loads (command sequences) into a NASCOM block and sends it via NASCOM to the TAC. The TAC would then transmit the OBC loads (command sequences) to the spacecraft.

GENERIC POCC ARCHITECTURE - RECOMMENDED REFINEMENTS

This part of the document describes a series of three refinements to the current generic architecture. The third refinement results in the recommended generic architecture. The recommended architecture, together with associated rationales, will form the basis of the rapid synthesis environment to be developed in the remainder of this task.

The refinements and rationales are presented in 5 subsections. The first section raises design issues concerning the current generic architecture. The second section aplies principles of object-oriented development to identify solutions to the issues raised. The third section summarizes the recommendations in the form of focused entity-relationship diagrams. Each of these diagrams focuses on a particular issue, and presents the current approach alongside the recommended approach.

The fourth section summarizes the alternatives in tabular form. For the tim[illegible] being, these tables constitute our database of architectural alternatives. The tables [illegible] provide a consise form in which to view the current architecure, the recommended [illegible] refinements, and various intermediate options.

The final section is a series of three configuration lists, similar in structure [illegible] which introduced the current generic architecture. The three lists represent the se[illegible] recommended refinements. As in the case of the current generic architecture, des[illegible] that represent intermediate phases, i.e., that differ from the ultimate (third) recomm[illegible] approach, are indicated in boldface.

ISSUES IN DEVELOPING POCC ARCHITECTURES

The following is a discussion of some of the issues involved in specifying a POCC-AP architecture. The list of issues was developed during our comparison of existing AP architectures and during our generation of a generic AP architecture. The issues are classified according to the major functional areas addressed. Typically, a designer's decisions concerning one of these issues will impact several of the subsystems that appear in an AP architecture. This impact can range from eliminating a subsystem from the architecture to modifying the functionality associated with a subsystem or modifying the internal structure of a subsystem. Our discussion of each issue presents the effects that alternative resolutions of the issue may have on the AP architecture.

Within some of the issue descriptions we make recommendations on which alternative should be pursued. Most of our recommendations are based on the following five design principles. These principles are common in modern design methodologies such as Object-Oriented Development:

o Separate real-time processes from non-real-time or intermittent processes.

o Isolate related activities in a single subsystem.

o Separate dissimilar or unrelated functions.

o Remove knowledge of a system's internals from the systems using that system.

o Isolate interfaces between the system and external entities.

3.1 AP Control

The issues associated with this functional area address how control flows through the AP to make the subsystems work in unison.

Distributing initialization to all subsystems. In some of the APs we looked at system initialization is performed in a single subsystem. For instance, in GRO initialization is performed in the Offline/History/ODB subsystem. It is our belief, however, that some initialization is being done by each of the subsystems, even though it is not shown in the SRS. The Display subsystem may initialize its data queues. Telemetry may initialize some of the TUT fields, and so on. We believe that it would be better to make this initialization process explicit in future APs. Each subsystem should have an internal initialization process that determines how the subsystem repsonds to the initialization directive, and the initialization directive should be broadcast to all subsystems. This design adheres more closely to the principles of object-oriented design, and it will allow new subprocesses to be added to a subsystem and initialized without having to change a centralized initialization process in some other subsystem.

Making all subsystems responsible for responding to their own directives. The actual control of the AP is not well-defined in the AP SRSs we reviewed. There appear to be two different ways for directives to be processed. Which path is taken is dependent upon the directive type: command or non-command. Command directives translate into spacecraft commands. They go directly to the Command subsystem where they are converted into spacecraft commands. The non-command directives are used to control the internal processing of the AP. How this is done can vary between systems. In GRO, the system appears to be controlled through the STOL subsystem. STOL resolves each non-command directive into the appropriate procedure call(s) in the AP subsystems. An alternative way to process non-command directives is to allow each subsystem to respond to its own directives (i.e., there is no single subsystem that interprets these directives into procedure calls). We would still need a subsystem to verify the existence of an issued directive, verify an issuers rights, and route the directive to the appropriate subsystem. This implementation has the nice property that it decouples the input of directives from their actual processing.

3.2 Operations Database

The issues associated with this functional area address how the items in the Operations Database are created, accessed and manipulated.

Centralizing access to the ODB. In the systems we looked at, the ODB is created by converting some raw specification of the ODB into a set of operating system files. These files are then accessed directly by the other subsystems. This means that the AP subsystems must incorporate an intimate understanding of the ODB's internal structure into their logic. Although this does make the system faster, it can make maintenance difficult. Any changes to the ODB will probably require detailed changes to one or more subsystems. We feel that this distributed access should be replaced by some form of centralized access that removes knowledge of the ODB's lowest level structures from the subsystems. Given today's high-speed processors and storage devices, it seems unlikely that these changes would seriously degrade the system's performance.

Supporting interactive maintenance of the ODB. In the COBE system, editing of the ODB is mentioned explicitly. This functionality should be included in all future APs.

Distributing ODB across subsystems. In Ford's AGS system, the ODB is partitioned according to the major subsystems in the system. Each subsystem contains the ODB information it uses in its processing. This approach seems to adhere more to the principles of object-oriented development than does the single centralized ODB. It does, however, risk degrading the maintainability of the ODB. One possible compromise would be to maintain a single ODB, but define individual views (i.e., subschema) for each subsystem. The subsystems would then only need to know about the data that pertains to their functionality. This approach may increase the time required to access the data. Another approach would be to use a central database during maintenance, and generate new subsystem databases from the single ODB each time the ODB changes. This approach may be unacceptable if the subsystems must modify their databases during their processing.

3.3 Operator Interface

The issues associated with functional area address how the operator and AP interact.

Combining the Operator Interface functions into a single subsystem. Currently, the control of the operator interface device(s) is spread across several subsystems. In GRO for example, the keyboard input is handled by the External Interface subsystem, but the screen output to the operator is handled by the Display subsystem. All control of the operator input and output devices should be combined into a single subsystem. This will isolate knowledge of the devices in a single subsystem, and it will support changes to the interface. This is particularly important given the likelihood that these devices will be the same (or at least overlap) in the future.

The actual processing of the operator input and the handling of AP output to the operator should also be combined into the same subsystem. This recommendation reflects the likelihood that both of these interfaces will be highly interdependent in the future. Previous operator input will determine how AP output should be displayed, and previous AP output will determine the options available to the operator.

Combining control of External Simulator and AP into a single Operator Interface. One trend that has been common in the Control Center arena has been to try to reduce the number of controllers required for each mission. One possibilty would be to combine the roles of AP operator and External Simulator operator. This seems to be consistent with the plan to move some control functions to the customer sites. The AP (and AP controller) would then be responsible for checking compatability between the customers' S/C commands. An advanced simulator would be an excellent way of doing this. Adding these responsibilties to the AP would dictate extensions the operator's AP control interfaces.

Knowledge-based Operator interface. As satellites and their payloads become increasing complex, the number of monitoring and control functions may become too large to be understood by a single individual. One way to to remove some of this load from the operator would be to incoporate some form of expert system into the AP. The expert system could perform many of the standard monitoring and control functions. The human operator would handle the less frequent but more challenging control tasks.

3.4 External Interfaces

The issues associated with this functional area address how the AP interacts with systems external to itself. These systems may include: TAC, DOCS, RUPS, TDRSS, STDN, GMT sources, CMS, FDF, the NCC, and/or the External Simulator. This section does not address how any of these systems might be controlled by or control the AP. These issues are discussed in other sections. Instead, this section discusses the actual linkage between these systems and the AP.

Centralizing network interface functions. In COBE and ERBS, Telemetry handles the routing of NASCOM blocks, and each subsystem interacts directly with the appropriate networks. Furthermore, the initial processing of NASCOM blocks is done in Telemetry. In GRO, there is an External Interface system, but its functions are limited to routing information between externals and subsystems. It does not perform any formatting, deformatting or processing functions. We believe that the reuse potential of the AP could be raised significantly by placing all knowledge of the networks in a single subsystem. This subsystem would then be responsible for deformatting incoming blocks and routing them to the appropriate subsystem, and formatting outgoing blocks and routing them to the appropriate external system. The subsystem might also have to translate AP internal block formats into external formats and vice versa. By doing this, the AP would easily support the replacement of networks and the addition of new networks.

Centralizing non-network interfaces to external systems. In addition to centralizing all network interfaces in a single subsystem, it would also help to centralize the interaction between the AP and other external systems in a single subsystem.

Currently, many of these external interfaces take place via tape transfers. In the future, these transfers will probably be via a network. By encapsulating these interfaces into a common subsystem, the AP will be able to support the sharing of these devices (e.g., networks and tape drives). However, this may require the interface subsystem to be configurable (i.e., it may be necessary to support switching of links between the AP and the external systems).

3.5 Device Control

The issues associated with this functional area address how the AP interacts with the peripherals it is connected to. These devices include: printers, storage devices, and special purpose devices (e.g., strip chart recorders). The control of devices associated with operator-AP interaction were discussed in Section 3.

Combining external device drivers (except Operator) into a common subsystem. In COBE, the event and line printers are controlled by STOL. In the original generic AP, event and line printers are controlled by Display. In ERBS, all subsystems may interact with the line printer directly. All of these approaches mean that several subsystems must have knowledge of how the printers (and other devices) are controlled. A better approach would be to centralize access to the external AP devices in a single subsystem. This approach has already been used in GRO's ODN. Centralization will make it easier to incorporate redundant devices (for failure protection) and to replace obsolete devices.

3.6 Command Processing

The issues associated with this functional area address how S/C commands are generated by the AP and how the S/C's response to these commands is verified.

Placing both OBC Load and Command verification in a single subsystem. In COBE, OBC memory comparisons are done in Display. In GRO, memory comparisons are done in Telemetry, but command verification is done in Command. In ERBS and the original generic, both types of comparisons are done in Command. The similarity between the two processes and the similar purposes of the two data types lead us to believe that this combination would probably reduce the complexity of the system by reducing the interactions between the subsystems, and would increase the potential for reuse.

Separating CMS Interface from Command. In the systems we looked at only GRO separates CMS Interface from Command. The rationale appears to be that CMS Interface handles the intermittent interactions between CMS and the AP, whereas Command handles the real-time generation of commands (both commands and OBC loads) and the verification of these commands. This separation should lead to increased performance of the system as a whole.

3.7 History Management

The issues associated with this functional area address how data is collected by the AP over time and how this data is used by the AP.

Separating Offline Processing from History. In the systems we looked there appears to be a common practice of combining History and Offline processing, or of subsuming one of these systems in another subsystem. In COBE and GRO, History and Offline are combined. In ERBS, there is no History subsystem. Instead, history is created by various subsystems but is processed by Offline. This practice seems to combine two subsystems with cognitively related goals but with very different processing requirements. History data is collected constantly throughout the AP's operation, whereas Offline processing occurs only intermittently. For this reason we advocate separating the two processes into distinct subsystems.

Telemetry replay controlled through History. How history data is stored and accessed should be controlled by a single subsystem. This isolates knowledge of the data's internal format in a single location. Telemetry data is one form of history data, and access to it should be controlled by the History subsystem. Therefore, the replay of telemetry data should be controlled by the History subsystem.

Telemetry replay done via Network Interface. The replay of data is intended to allow the AP to reenact previous S/C passes. In COBE, replay is initiated in History, but it is fed into Telemetry. This does not seem to be consistent with the idea of using replay to simulate a previous pass, since normally data enters the AP through the networks. We propose sending replay data through the Network Interface instead of through other subsystems.

3.8 NCC Data

The issues associated with this functional area address how the AP interacts with the NCC to process NCC requests and status information, and to initiate inquiries concerning NCC status.

NCC Processing handles all interaction with NCC. Interactions between the AP and the NCC should be handled by a single subsystem. This interaction should not take place via other subsystems except where absolutely necessary. In COBE and ERBS, NCC messages are routed through Telemetry and Command. This appears to be because these systems handle incoming and outgoing NASCOM data. In our original generic AP, we have eliminated the link between NCC Processing, and Command and Telemetry because we isolated interfaces to the networks in Network Interface.

3.9 Simulation

The issues associated with this functional area address the topic of simulation. Simulation involves simulating those systems that are external to the AP and that are critical to assessing the AP's real-time performance. These systems include the spacecraft and the NCC. Simulation may be done within the AP, by external simulator, or both.

Providing internal simulation. In most of the systems we looked at there does appear to be telemetry simulation occuring in the AP. It is usually done as part of the Telemetry subsystem. This placement seems to be mixing different types of processing in the same subsystem. An alternative approach would be to make the simulator a separate subsystem. The Telemetry subsystem would then handle only the processing of incoming telemetry data, and the Simulator subsystem would perform the simulation.

Providing direct links between the AP and the External Simulator. In COBE, all interaction between the AP and the External Simulator is through the TAC. In ERBS, communication with the external simulator can be configured either way, through TAC or via a direct link. In our original generic AP, the External Simulator is controlled either by the Operator (locally or remotely) or through the Command subsystem. Furthermore, the External Simulator's output enters the AP through the Network Interface directly, instead of through the TAC. This approach of allowing the AP to interact directly with the External Simulator for both control and data exchange provides a lot of flexibilty in controlling the simulator, and its makes the simulator identical to the spacecraft in terms of data flow.

3.10 Teleoperations

The issues in this section do not a single functional area. Instead, they represent changes in how spacecraft commands and data will be handled in the future. These changes may impact several AP subsystems.

Distributing some command functions to the customers. According to the CDOS directives, the actual creation of S/C (i.e., payload) commands will be initiated at customer sites instead of in the AP as they are now. This change will probably increase the functionality of the AP instead of reducing it. For instance, the AP will have to assure consistency and handle scheduling of the customer's commands, and it will have to verify each user's rights to issue received commands.

Distributing payload control and data processing to the customers. In future space missions there will be multiple payloads on each spacecraft. Customers will be allowed to issue commands to their payloads and to directly access the data produced by them. However, all such interactions will involve the use of shared spacecraft resources (e.g., power, communications devices, etc.). The AP will have to have some control over these shared resources, particularly in spacecraft emergencies. In order to avoid damaging the payloads, the AP will need to know how altering the status of any of these resources will affect the payloads. The AP may also need to coordinate these changes with the customers.

3.11 Subsystem Design

The issues included in this section do not address any specific functional area. They address more general design concerns about what items should be included in a subsystem and how these items should be organized.

Combining Subsystems. Many of the above suggestions have implications concerning what the subsystems of an AP should be. Some, such as the basic principles listed in the introduction, only make general suggestions about when to group and when not to group functions into a subsystem. We mention this issue here because the way in which a system is partitioned can have a dramatic effect on its maintainability.

Encapsulating Data. One of the best ways to increase the maintainability of a system is to make all interfaces within the system explicit. One of the best ways to make interfaces obscure is through data access. In many systems developed in the past, a lot of data has been made global within the system, and has been modified and read by several subsystems. This can make it difficult for a maintainance person to understand exactly what the links between subsystems are and how the links control the processing in the system. To avoid this problem, we suggest eliminating the use of global data (at least at the system level). Instead, each data item should placed within the primary system that modifies or accesses it. We call this process data encapsulation.

Levelling of functionality in subsystems. As APs become more complex, the number of functions appearing at the top level of a subsystem will increase. This is likely to make the subsystem less understandable, and therefore less maintainable. Our suggestion is that when a subsystem has more than seven top-level items, the developer should introduce a new level of abstraction into the subsystem. This involves grouping functions that act on the same or similar data into a new component. These new functional groups then appear at the subsystem level, and the original functions become sub-functions of them. The trend toward more levels of abstraction to control complexity can be seen in the GRO SRS.

OBJECT-ORIENTED PRINCIPLES AND AP ARCHITECTURES

In developing our generic POCC-AP architectures we used the five basic principles described in the section on AP architecture issues. These principles give guidelines as to when functions should be placed in the same subsystem and when they should not. The results of applying these principles to existing AP architectures are summarized in our generic architectures. The APs produced are very similar to APs of the past.

In this section we consider what effect adding the tenets of object-oriented development to our list of principles would have on AP architectures. We begin with a discussion of what these tenets are. These tenets are not replacements for our original principles; they are additions to them.

Developing Good Abstractions

The fundamental principle underlying object-oriented development is that system components should be grouped according to the natural abstractions they address. In defining good abstractions it is necessary to consider all of the following issues:

- o *Application domain entities:* At the upper levels of a system architecture all of the components (e.g, subsystems) should correspond to either real-world entities in the problem domain or major types of information being manipulated by the system. An example of the former is the Operator. An example of the latter is spacecraft commands.

- o *Completeness of the abstraction:* When an abstraction is being defined, it is important to identify all of the responsibilties of the abstraction, both those directly stated in the requirements and those implied by the requirements or previous domain knowledge. The more complete the abstraction, the less likely it is to need changes in the future.

- o *Coupling between components:* The coupling between components (i.e., functions and data) can be a good indication of what items should be grouped together. In general, items in the same object should be more tightly coupled to each other than they are to items in other objects.

- o *Complexity in the system:* The number of top-level items appearing in a system or subsystem should be small (< 9). When this is not the case, it typically indicates that some higher level abstraction exists under which some of the items should be grouped.

- o *Impact of the future:* When considering the above issues, it is frequently necessary to consider how the system will change in the future. For example, two items may not currently be tightly coupled, but if it is likely that they will be in the future, they should be grouped. Planning for the future will make the system easier to maintain.

Controlling the Coupling Between Subsystems

In general, the coupling between objects should be kept as low as possible. This increases their potential for reuse. The subobjects (and items) of an object should be more closely coupled to each other than they are to objects outside the given object. The interface routines for an object should be coupled only to the data internal to the object; they should not access data outside the object. They may of course use routines outside the object.

Data coupling. In object-oriented development, it is considered bad practice to allow a function in one object to access the data from another object, particularly if this data is modified. First, it suggests that we do not have well-defined abstractions on which to base the objects. Secondly, it results in an implicit interface between the objects. This can have negative effects on the maintainability and reusability of the code. The practice of providing global data regions was common in many system developments of the past.

The best way to support the sharing of data between objects is to provide call-by-value access functions in the object containing the data. In short, we should not allow access directly to the data from outside the object. If access is necessary, a copy of the data should be made and then sent to the accessing routine.

However, sometimes we really do want to share data between objects. For instance, when we pass a NASCOM block from one part of a POCC to the other, we don't want to copy the block and then send it; we want to send the actual block. This can be done by encapsulating the data within a smaller object and then passing the smaller object between the larger objects. The encapsulated data may only be accessed by the routines in the object. In this way a well-defined contract is established between the two accessing objects. Both objects know exactly what are the limits on the way the information may be changed by the other.

Real-time processing may (in extreme cases) necessitate sharing data between objects.

Sharing common functions. Unlike sharing data, object-oriented development is not adverse to the idea of sharing functions. The more a routine can be reused in an application the better. We do this all the time without even knowing it. For example, every numeric operation in a program is really a reuse of some compiler supported routine. Other common candidates for reusable routines include database management routines, device drivers, and I/O routines. When the developer identifies a set of commonly used routines he should place them in a service package so they may be accessed by other parts of the system and so there is only one copy of the item. Care must be taken, however, when reusing routines that have side effects (i.e., routines that change the value of a data item or a device external to themselves), and/or are dependent on the value of a data item external to themselves. When this is the case, it may be impossible to predict the results of calling the routine from other parts of the system. The correct way to handle such routines is to encapsulate them along with the data (or devices) they act upon into a single object which is accessed by other parts of the system. If the routines are intended to act on different items in different parts of the system, then it may be desirable to group the routines into a template and instantiate the template for different parts of the system. In Ada this is done using generics.

Sharing data types. Object-oriented development encourages the sharing of data types throughout a system. However, it does require the type itself be hidden within an encapsulation that includes the routines that may operate on instances of the type.
vice packages

Applying OOD to POCC-AP Architectures

The following is decription of how applying the OOD tenets will effect the POCC-AP architectures we have developed.

Grouping top-level subsystems. If a developer were to use the initial set of recommendations from Section $$$, he would generate an AP architecture with more than fifteen top-level subsystems. Such an architecture is difficult to understand, even using a simple entity-relationship model. We, therefore, went back to our original statements of how an AP works and looked for higher-level component abstractions. By doing this, we were able to lower the number of top-level subsystems from more than fifteen to seven. We also identified several high-level relationships that we had not considered before.

Unifying operator functions. All of the functions that help the operator interface to the AP should grouped into a common subsystem. In our current generics, the two directions in this interface are handled by different subsystems (Operator Input and Display) with little interaction between the two. However, as the operator interface becomes more complex, it will become necessary to employ advanced display management techniques in order to avoid overwhelming the operator. This will almost certainly involve basing the presentation of AP output on recent operator input and limiting the availabilty operator input options based on recent AP output. In short, the handling of operator input will be tightly coupled with the handling of AP output. Therefore, the Display and Operator Input subsystems should be combined into a single subsystem.

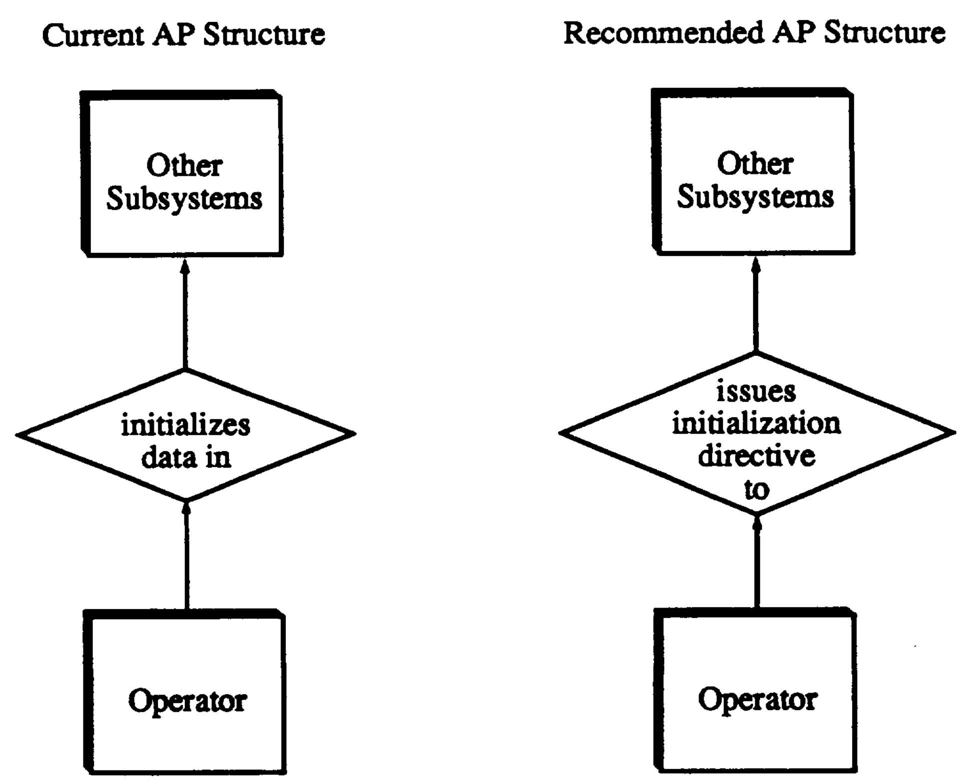
Moving NCC from Operator to Network. It is possible to view the handling of AP-NCC interaction as being part of the operator's role or part of the interface between the AP and the network(s). In our generic architectures it could be placed, therefore, in either the Operator or Network subsystems. At the present time, there is no strong data coupling between the routines associated with the AP-NCC interface and either of the subsystems. Thus it would seem rather arbitrary as to where the routines should go. But if we look at possible future additions to the AP we can forsee the need for several AP-network controller interfaces. We would like to place all such interfaces in a common subsystem due to their very similar goals. Many of these interfaces may require detailed knowledge of the networks themselves. We therefore advise placing the routines (and data) associated with the AP-NCC interface in the Network subsystem.

Removing data from ODB. As we pointed out above, objects should not share access to data. This includes data in a central database. The best way to ensure that this is the case encapsulate the data items within the subsystems which access them. We, therefore, advise against using a single central ODB in future APs. Instead, the data items should encapsulated within the subsystems that access them. If data must be accessed by several subsystems, it should be encapsulated in a separate object and then accessed through the object's interface. The developer may still implement each of the data items using database technologies, but the databases will have to be maintained separately.

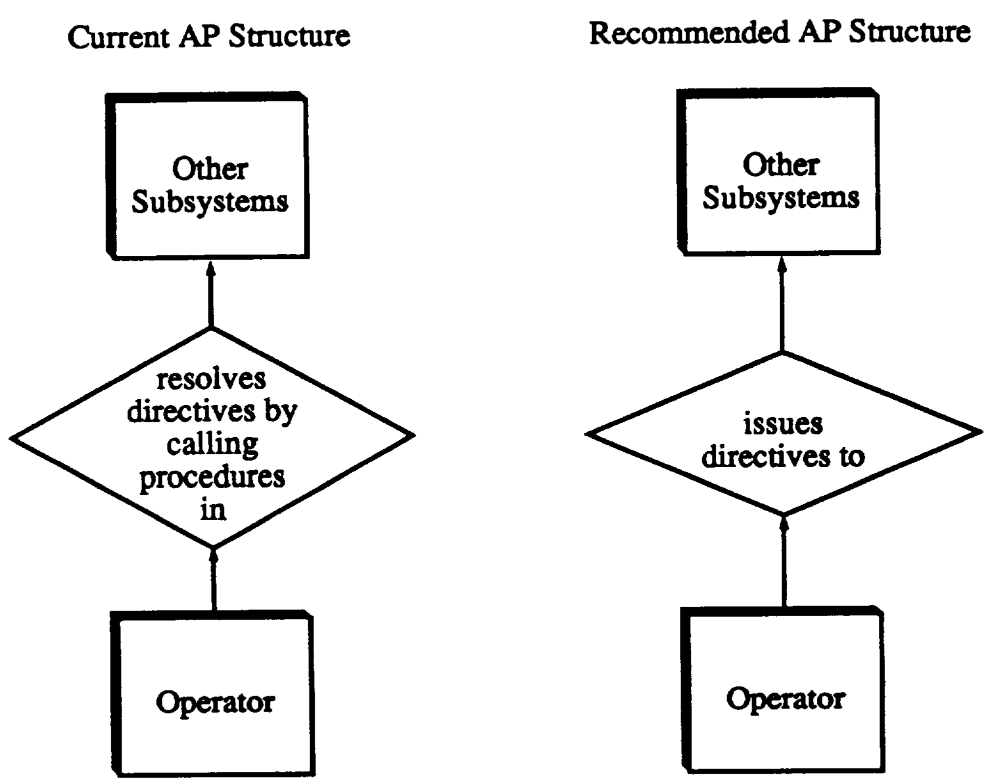
Unifying History and Offline. Originally we separated the routines managing history data and for producing reports based on this data into two separate subsystems, namely History and Offline. This was done because the processing requirements for each type of routine are very different. History management is basically a real-time process, whereas the generation of reports is predominantly non-real-time. We still feel that this distinction is valid, but we now recognize the both sets of routines will have to have detailed knowledge of the history data's structure. Therefore, the History and Offline subsystems should be combined into a single subsystem. The associated with history management and report generation may still be separated within the new subsystem, and the data that they share may encapsulated in a shared object.

RECOMMENDED ARCHITECTURE REFINEMENTS

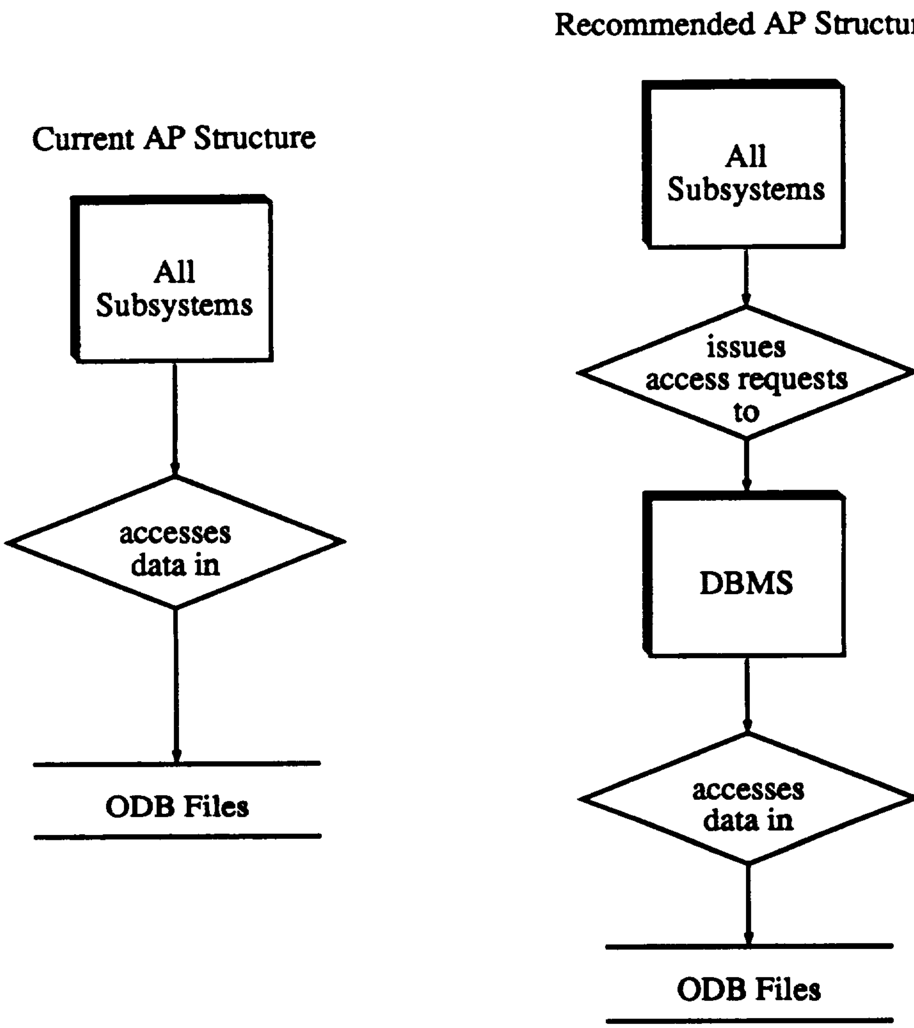
ENTITY-RELATIONSHIP VIEW



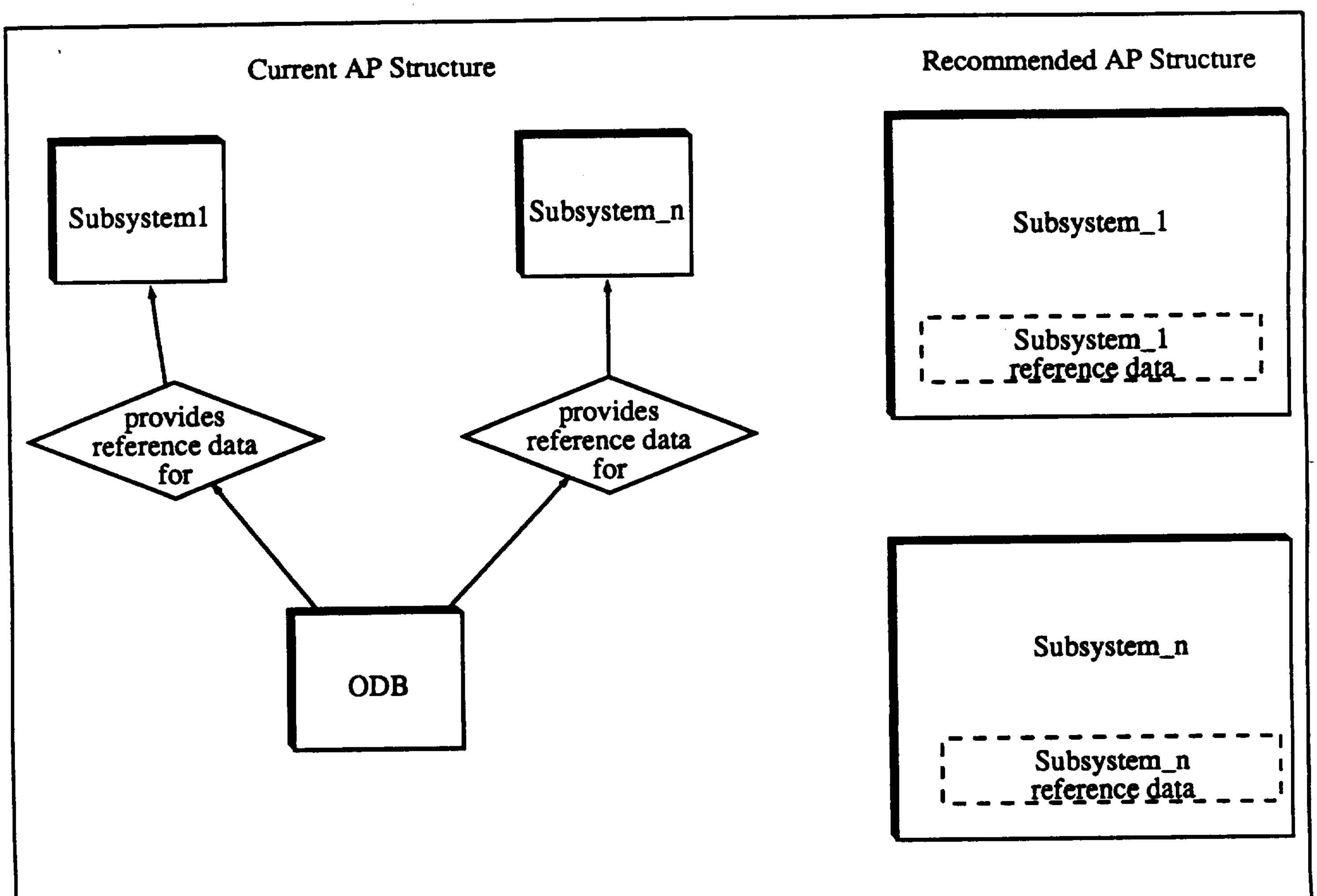
Distribute Initialization to all Subsystems



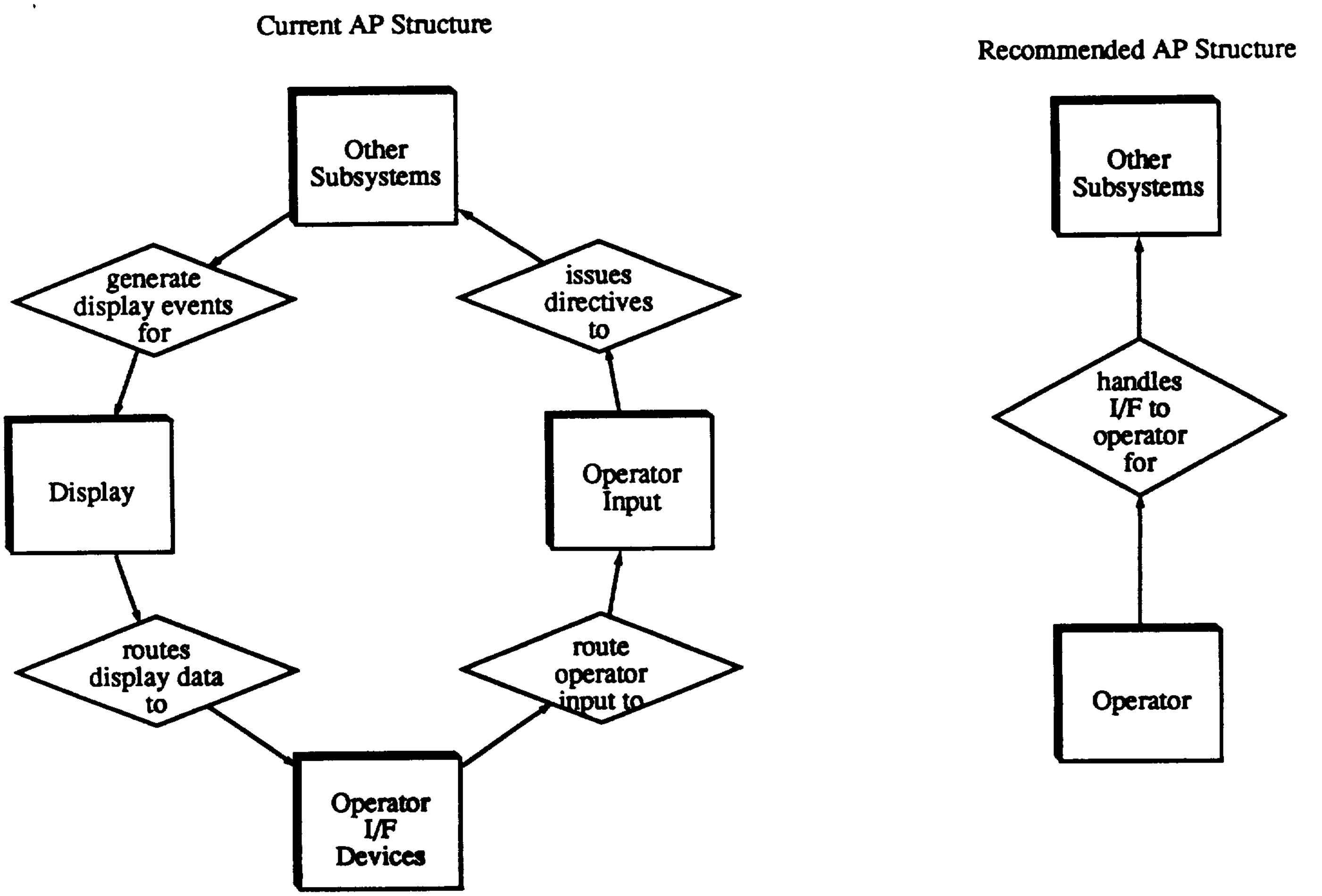
Make Subsystems Responsible for Directive Processing



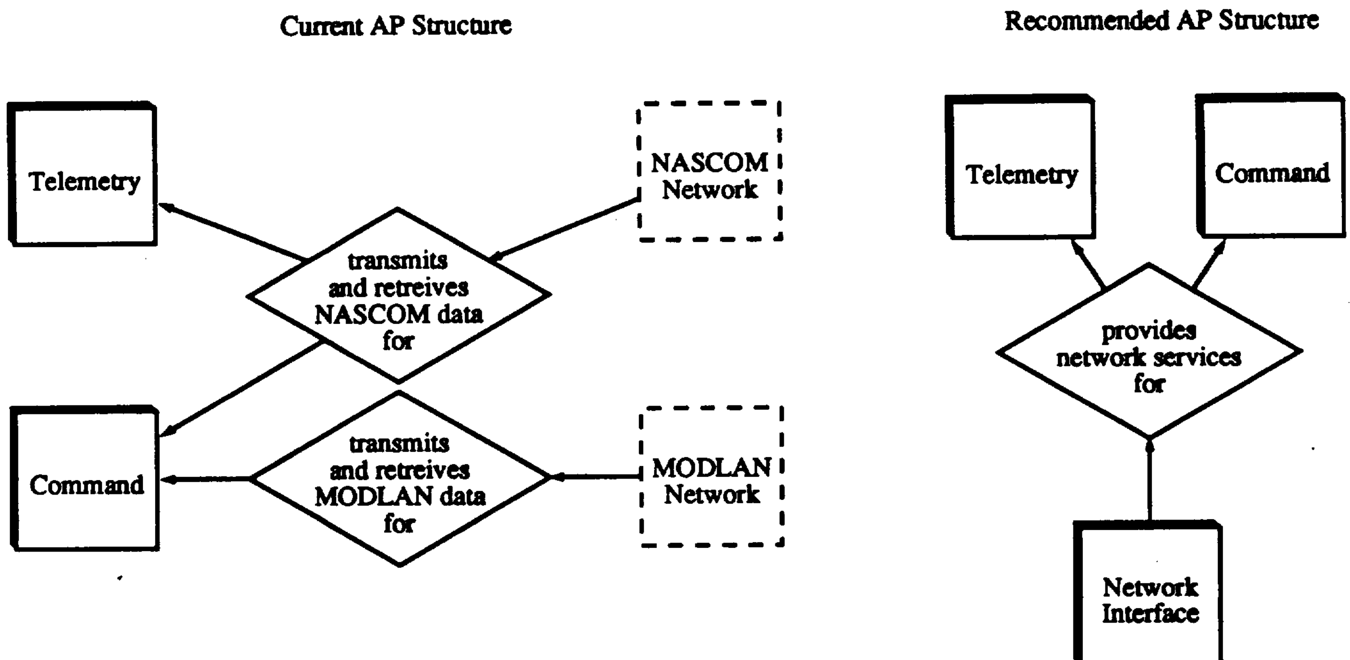
Control Access to ODB



Distribute ODB Across Subsystems



Combine Operator Interface Subsystems



Central Network Interface

Database of Reusable POCC Architectures

Configuring a particular POCC architecture requires supplying answers to questions raised in the preceding discussion. These decision points have been summarized in the matrix shown in Figure ____. This matrix is machine processable and can be combined with a dialog-based specification tool to define a particular architecture. The "POCC-AP Pieces" matrix shows all of the subsystems and functions that can assume other that their pre-assigned positions in the Generic POCC-AP architecture previously defined in Section ___. Changing any of these items, results in a changed architecture. The first section shows a complete list of the subsystems of the generic architecture, whether configurable or not. The second section shows only those functions which are configurable. All other functions are assumed to reside permanently within their designated subsystems and are therefore not configurable, and not included in the matrix.

The columns of the matrix have the following meanings:

- "Combinable with" shows any other subsystem that the current subsystem may be combined with for a specific architecture. No assumption is made regarding the name of the surviving subsystem.

- "Movable From, To" indicates the default location of a function and one or more alternative destinations to which it may be moved.

- "Optional", if selected, means that the indicated subsystem, function or interface may or may not be included. In configuring a specific architecture, this choice must be made.

- "Distributed/Centralized", if selected, means that a choice must be made between distribution and centralization of the subsystem or function.

Figure ___

Configurable POCC-AP Pieces

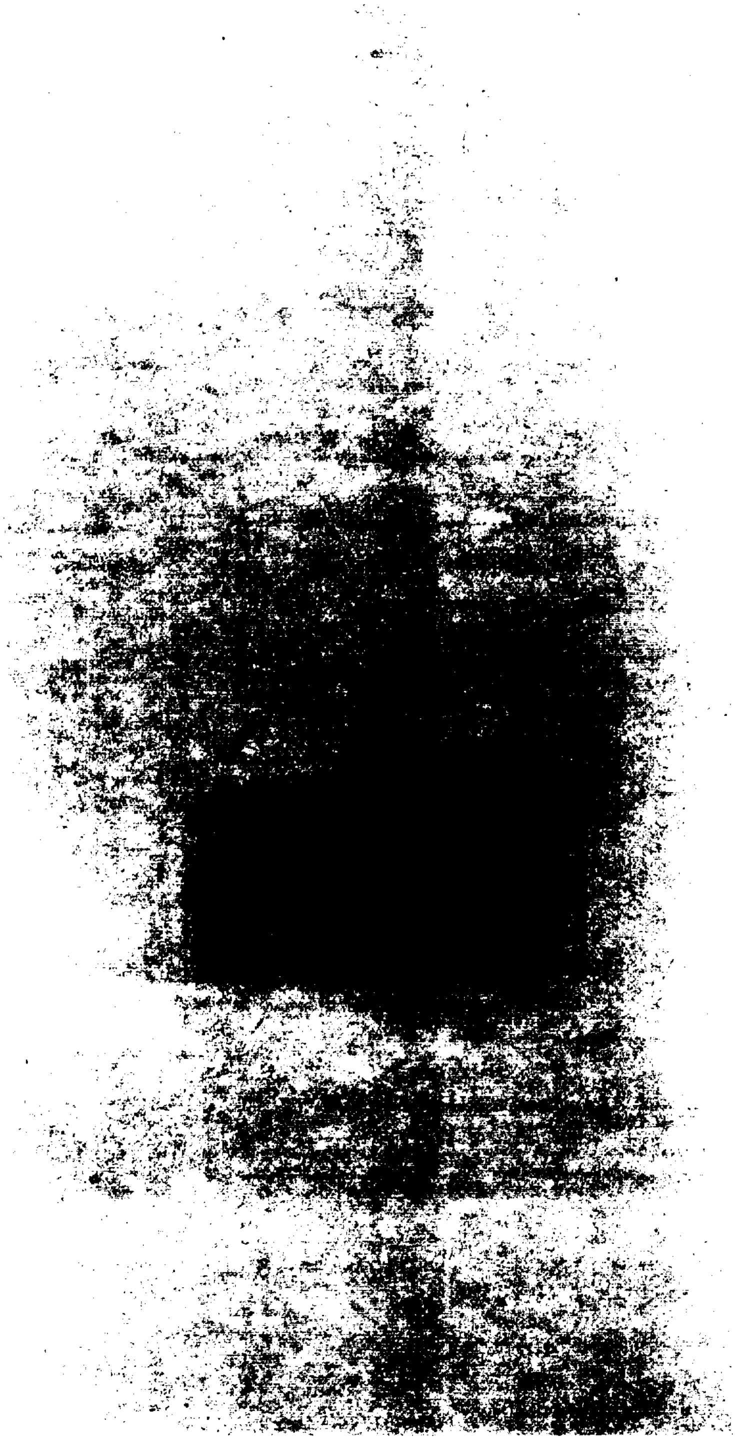
Subsystems	Combinable With	Optional
NI Network Interface		
OF Offline	HI,DB	
TE Telemetry		
OI Operator Input	DI	
DI Display	OI	
HI History	OF,DB	
ES External Simulator		X
IS Internal Simulator		X
NP NCC Processing		
CI CMS Interface	CO	
DB Data Base		

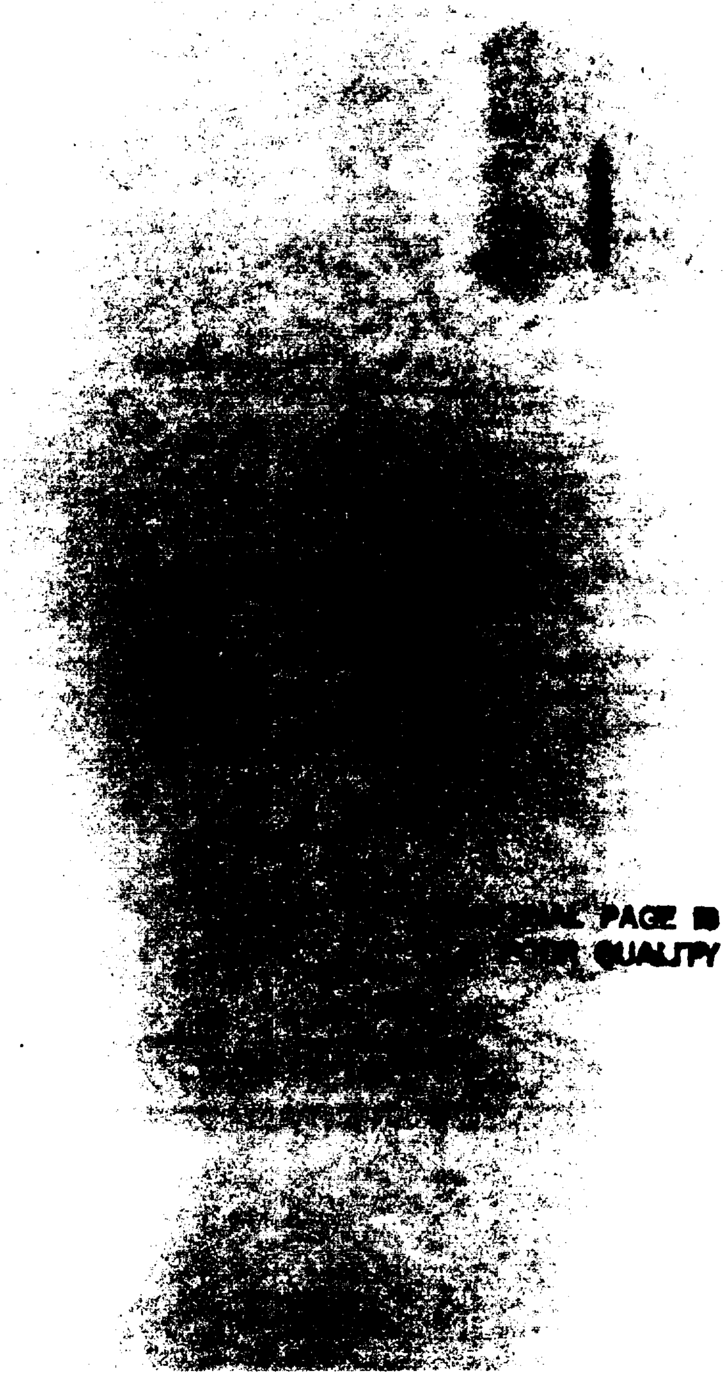
Functions	Movable From	Movable To	Optional	Distributed/ Centralized
Initialization				X
Database Access				X
Database Files (or Views)				X
Internal Simulator	TE	IS	X	
External Simulator Control	ES	OI		
Directive Processing				X
Customer Directive Input			X	
Single Interface for Operator Devices			X	
Device Control				X
Knowledge Based UIF			X	
Load/Image Verification	CO	DI,TE		
Telemetry Replay Generation	OF	HI,NI		
Non-Network Interfaces				X
Network Interface				X

Interfaces	Optional
TDRSS	X
STDN	X

RECOMMENDED ARCHITECTURE REFINEMENTS

CONFIGURATION LISTS





POCC AP--RECOMMENDED GENERIC ARCHITECTURE CONFIGURATION (STAGE 1)

Subsystems

Offline	Separate
Operator Input and Display	Separate
History	Separate
External Simulator	Yes
CMS Interface	Separate

Functions

Initialization	Distributed
Database Access	**Distributed**
Database Files	Centralized
Internal Simulation	Separate Subsystem
External Simulator Control	**Local and Remote (AP) Operator**
Directive Processing	Distributed
Customer Directive Input	**No**
Single Interface for Operator	Yes
I/O Devices	
Device Control	Centralized
Knowledge Based UIF	**No**
Load/Image Verification	In Command
Telemetry Replay	In History
Non-Network Interface	Centralized
Network Interface	Centralized

POCC AP--RECOMMENDED GENERIC ARCHITECTURE CONFIGURATION (STAGE 2)

Subsystems

Offline	Separate
Operator Input and Display	Separate
History	Separate
External Simulator	Yes
CMS Interface	Separate

Functions

Initialization	Distributed
Database Access	Centralized
Database Files	Centralized
Internal Simulation	Separate Subsystem
External Simulator Control	Remote (AP) Operator only
Directive Processing	Distributed
Customer Directive Input	Yes
Single Interface for Operator	Yes
I/O Devices	
Device Control	Centralized
Knowledge Based UIF	NO
Load/Image Verification	In Command
Telemetry Replay	In History
Non-Network Interface	Centralized
Network Interface	Centralized

POCC AP--RECOMMENDED GENERIC ARCHITECTURE CONFIGURATION (STAGE 3)

Subsystems

Offline	Separate
Operator Input and Display	Separate
History	Separate
External Simulator	Yes
CMS Interface	Separate

Functions

Initialization	Distributed
Database Access	Centralized
Database Files	Centralized
Internal Simulation	Separate Subsystem
External Simulator Control	Remote (AP) Operator only
Directive Processing	Distributed
Customer Directive Input	Yes
Single Interface for Operator	Yes
I/O Devices	
Device Control	Centralized
Knowledge Based UIF	Yes
Load/Image Verification	In Command
Telemetry Replay	In History
Non-Network Interface	Centralized
Network Interface	Centralized